Titre: Title:	Exploration architecturale pour la conception de processeurs réseaux basée sur l'utilisation de processeurs configurables
Auteur: Author:	David Quinn
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Quinn, D. (2003). Exploration architecturale pour la conception de processeurs réseaux basée sur l'utilisation de processeurs configurables [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7295/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7295/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

EXPLORATION ARCHITECTURALE POUR LA CONCEPTION DE PROCESSEURS RÉSEAUX BASÉE SUR L'UTILISATION DE PROCESSEURS CONFIGURABLES

DAVID QUINN
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 2003

Your file Votre référence
ISBN: 0-612-90855-0
Our file Notre référence
ISBN: 0-612-90855-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

EXPLORATION ARCHITECTURALE POUR LA CONCEPTION DE PROCESSEURS RÉSEAUX BASÉE SUR L'UTILISATION DE PROCESSEURS CONFIGURABLES

présenté par : QUINN David

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

MME NICOLESCU Gabriela, Doctorat, président

M. BOIS Guy, Ph.D., membre et directeur de recherche

M. CHAMBERLAND Steven, Ph.D., membre et codirecteur de recherche

M. PIERRE Samuel, Ph.D., membre

Remerciements

Je tiens d'abord à remercier mon directeur de recherche Guy Bois pour m'avoir donné l'opportunité de travailler sur ce projet et pour m'avoir accordé toute sa confiance. Je remercie aussi mon codirecteur Steven Chamberland pour sa grande accessibilité et ses commentaires constructifs. Je me vois également fort reconnaissant envers le CRSNG pour la bourse qu'il m'a octroyée. Son aide financière m'aura permis de me consacrer, durant ces dernières années, corps et âme à la recherche.

Mes remerciements vont également à l'endroit de l'école Polytechnique, plus spécifiquement au groupe Circus, pour son environnement de travail chaleureux et des plus stimulant. Je remercie d'ailleurs tous mes collègues qui ont participé de loin ou de près à la réalisation de ce projet. Je tiens entre autre à souligner l'importante collaboration de Bruno Lavigueur qui a travaillé ardemment à l'intégration du Xtensa dans StepNP. Je tiens aussi à remercier la formidable équipe de *STMicroeletronics SoC platform automation group* d'Ottawa pour leur don de StepNP, leur soutien technique et toutes leurs bonnes idées qui m'auront aidé à surmonter plusieurs embûches tout au long de ce projet.

Je désire finalement remercier ma conjointe Kim Blouin pour ses nombreux encouragements. Elle m'aura surtout permis de me changer les idées en dehors du travail en m'accueillant bras ouverts à toutes heures du jour et de la nuit. Je profite d'ailleurs de l'occasion pour saluer affectueusement ses deux chats : Jules et Juliette, tous deux très attachants.

Résumé

La démocratisation de l'Internet a révolutionné le marché des télécommunications des dernières années. Produisant un intérêt sans égal, on a vu la demande en largeur de bande littéralement exploser et un accroissement important de la diversité des services. Pour répondre à cette demande, l'infrastructure des réseaux dorsaux a dû être repensée. Pour la conception de routeurs à haute performance, les solutions actuelles ont recours aux processeurs réseaux. Ces SoC multiprocesseurs utilisent alors des processeurs embarqués d'usage général ou des ASIP pour l'exécution des tâches réseaux. Bien que très flexibles, les plates-formes exclusivement logicielles ne satisfont malheureusement pas toutes les contraintes. Un nombre impressionnant de coprocesseurs doivent alors être ajoutés pour atteindre les performances désirées, résultant non seulement en un accroissement de la complexité des designs, mais aussi de leur temps de développement et d'apprentissage.

Toutefois, l'arrivée récente des processeurs configurables pourrait venir simplifier et accélérer la conception des processeurs réseaux, sans en dégrader la performance. En effet, autre leur grande configurabilité, ces processeurs offrent la possibilité d'ajouter des instructions spécialisées à leur jeu de base. Engendrant dans bien des cas des gains similaires aux coprocesseurs, les nouvelles instructions sont aisément intégrées à l'architecture et la génération des outils de développement logiciel est automatisée.

D'abord, afin d'évaluer adéquatement les éventuels bénéfices de l'utilisation des processeurs configurables dans la conception des processeurs réseaux, nous proposons une méthodologie de partitionnement mieux adaptée à leur utilisation. Lorsque le logiciel ne produit pas les gains escomptés, une nouvelle alternative est alors privilégiée à l'usage d'accélérateurs matériels. En effet, compte tenu de leurs avantages, l'ajout d'instructions spécialisées est maintenant envisagé avant celui de coprocesseurs.

Par la suite, nous nous sommes dotés d'une plate-forme permettant l'exploration rapide d'architectures. Notre choix s'est alors arrêté sur StepNP, puisque celle-ci était disponible et possédait déjà une bibliothèque de composants pour les processeurs réseaux ainsi qu'une suite d'outils pour le débogage et l'analyse de performance. Basée sur SystemC, StepNP facilite la conception et l'intégration de modules. Toutefois, comme elle ne comportait aucun processeur configurable, nous y avons intégré un ISS du Xtensa de Tensilica. Une plate-forme monoprocesseur a alors pu être développée sans aucune difficulté.

Finalement, deux applications réseaux ont été réalisées. Très répandue pour les communications de niveau 3 et effectuant une grande variété de tâches réseaux, une implémentation d'IPv4 a d'abord été choisie. Puis, pour combler l'absence d'algorithmes complexes, nous y avons ajouté le cryptage DES.

L'approche proposée a alors été appliquée pour chacune des deux spécifications. Les résultats de cette recherche démontrent que l'ajout d'instructions spécialisées est aisé et peut effectivement accélérer significativement certaines tâches réseaux, sans pour autant complexifier l'architecture d'un design.

Abstract

The democratization of the Internet has revolutionized the telecommunication market of the last years. Bearing an interest without equal, this trend has caused a bandwidth explosion and an important increase in the diversity of the services. To match this demand, the infrastructure of the backbone networks had to be reconsidered. For the design of high performance routers, network processors are used. These multiprocessor SoCs use embedded general-purpose processors or application specific instruction processors for network processing. Although very flexible, platforms exclusively software do not satisfy all the constraints. A huge number of coprocessors must then be added to reach the specified performances, resulting not only in an increase of the designs complexity, but also in their development and programmers training time.

However, the emergence of configurable processors could overcome these problems by simplifying and accelerating the design of network processors without reducing their performance. By adding to them specialized instructions to match specific tasks, configurable processors can produce interesting gains similar to hardwired solutions. Moreover, new instructions are easily integrated into the processor architecture and the software development tools are automatically generated.

Firstly, to adequately evaluate the advantages of the configurable processors in the design of network processors, we proposed an improved partitioning methodology, adapted to their use. In our approach, when the software does not produce the requested performance, a new alternative is then privileged to the old one. Taking into account their advantages, the use of specialized instructions is now considered before that of coprocessors.

Thereafter, we chose the StepNP platform for fast architectural exploration. Since it was available and already includes components libraries for network processors and a tool set

for debugging and performance analysis. Based on SystemC, StepNP eases the design and the integration of new modules. However, because it had no support for configurable processors, we integrated into it the Tensilica's Xtensa ISS. A monoprocessor platform has then been easily developed.

Finally, we developed two network applications. Largely used for level 3 communications and consisting of different network tasks, an Ipv4 implementation was first selected. Then, to fill the lack of complex algorithms, we added the DES–CBC encryption algorithm.

We applied the proposed approach to each specification. The results of this research show that using specialized instructions is easy and can significantly improve some network tasks without increasing the design complexity.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

ADSP	Application Domain Specific Processor
AH	IP Authentication Header
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIP	Application Specific Instruction Processor
ATM	Asynchronous Transfer Mode
CBC	Cipher Block Chaining
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DES	Data Encryption Standard
DMA	Direct Memory Access
ESP	Encapsulating Security Payload
FCS	Frame Check Sequence
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
ICMP	Internet Control Message Protocol
IDEA	International Data Encryption Algorithm
IP	Internet Protocol
IPsec	Internet Protocol for security
IPv4	Internet Protocol version 4
ISO	International Standards Organisation
ISS	Instruction Set Simulator
LAN	Local Area Network
NAT	Network Address Translation
NoC	Network-on-a-Chip
OC	Optical Carrier (OC-1 = 51.84 Mbps)

OSI	Open System Interconnection
QoS	Quality of Service ou Qualité de Service
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SDP	Serial Data Processor
SoC	System-on-a-Chip
SONET	Synchronous Optical NETwork
SPD	Simple Packet Device
SSH	Secure SHell
StepNP	System-level Exploration Platform for Network Processors
TCP	Transmission Control Protocol
TIE	Tensilica Instruction Extensions
TTL	Time To Live
UAL	Unité Arithmétique et Logique
UCT	Unité Centrale de Traitement
WAN	Wide Area Network
XLMI	Xtensa Local Memory Interface

Liste des annexes

Introduction

Jusqu'à la fin des années 90, la grande majorité des routeurs reposaient sur des architectures similaires à celle des ordinateurs personnels [AGER99]. Presque toutes les tâches reliées au traitement de paquets (incluant la gestion des files d'attentes et des tables de routages) étaient réalisées par une unité centrale de traitement (UCT). De telles architectures avaient l'avantage d'être facilement adaptées aux nouveaux protocoles et aux changements rapides des besoins du marché. Un exemple classique et toujours pertinent est la ligne de produits 2500 de Cisco [CISC03]. Composé d'un seul processeur centralisé pour l'exécution des tâches réseaux (placées dans une mémoire RAM non volatile), ce produit vient avec différentes interfaces dépendamment des besoins. Bien que peu performant, une grande variété de configurations est aussi disponible rendant cette gamme de produits très flexible. La Figure 1 présente l'architecture traditionnelle basée sur l'utilisation d'un processeur central. Toutefois, pour la réalisation des routeurs d'infrastructure, l'explosion récente de la demande en largeur de bande et l'accroissement de la complexité des services requis ont surpassé le potentiel de ces architectures.

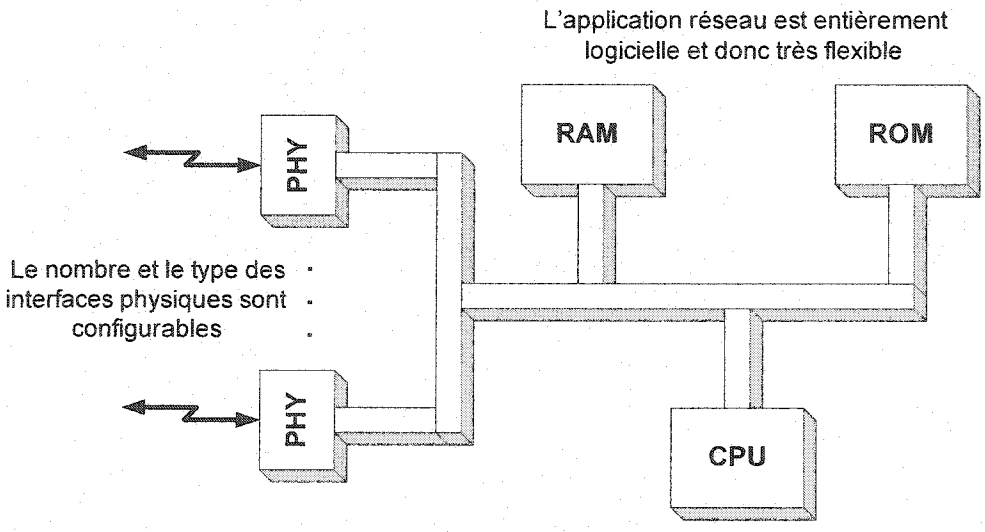

Figure 1 : Architecture des premiers routeurs

D'abord, la démocratisation de l'Internet a révolutionné le marché des dernières années. Produisant un intérêt sans égal, on a vu le nombre de clients croître significativement, où chacun réclame de plus en plus de bande passante. De plus, requérant des besoins davantage diversifiés (tels le transport de la voix sur IP (*Internet Protocol*) ou encore les réseaux privés virtuels), ces clients pressent les fournisseurs de service à déployer de nouvelles technologies et à repenser l'infrastructure de leurs réseaux. Résultat : le marché actuel se caractérise par une course féroce pour de plus grandes bandes passantes et les compétiteurs tentent de se différencier par les services offerts.

Deuxièmement, bien que la fibre optique satisfasse allègrement la demande en transport de l'information au niveau de la couche physique, un problème persiste au niveau du routage des paquets dans les réseaux dorsaux. Le traitement, d'une complexité accrue, doit être accompli sur un plus grand nombre de paquets et surpasse de beaucoup la capacité croissante des processeurs d'usage général. Comme ces architectures ne sont pas extensibles, un besoin urgent pour le développement de nouvelles approches et de nouvelles technologies dans la conception de plates-formes multiprocesseurs spécialisées à haute performance est apparu.

Heureusement, l'avènement récent des SoC (*System-on-a-Chip*) apporte une solution très intéressante aux besoins propres du domaine des applications réseaux. Ils permettent la fabrication de processeurs réseaux beaucoup plus performants, en offrant la possibilité d'intégrer sur une même puce un ensemble de processeurs et coprocesseurs spécialisés, une structure de mémoire avancée ainsi que des interconnexions et des interfaces hautes vitesses.

Problématique

Le style des processeurs réseaux commerciaux comprend tout le spectre des architectures, s'étendant des solutions complètement logicielles aux architectures presque entièrement matérielles. La configuration d'une architecture peut dépendre du débit visé, du type de

traitement effectué ou encore des contraintes du marché. Bien que les SoC fournissent le moyen, un défit majeur demeure en la grande complexité de ces systèmes. Des outils assistant efficacement les concepteurs dans l'exploration de ce vaste espace de solutions deviennent une nécessité pour rencontrer les contraintes du temps de développement restreint.

Plusieurs outils tentent déjà d'automatiser ou du moins faciliter le design des SoC. Du côté commercial, on retrouve entre autres Cadence VCC [CADE03] et CoWare N2C [COWA03]. Ce dernier propose une suite d'outils et une méthodologie de développement de plates-formes complexes basées sur l'utilisation de processeurs embarqués. Il utilise un flot de co-design logiciel/matériel semi-automatique, offrant des mécanismes pour la spécification et le raffinement de blocs de propriété intellectuelle, ainsi que pour la réutilisation d'éléments existants. De plus, CoWare intègre la technologie LISATek pour le développement de processeurs spécialisés. Une description dans un langage propriétaire doit d'abord être réalisée. Par la suite, tous les outils nécessaires sont générés automatiquement (simulateur, compilateur, débogueur, etc.). Cette même description peut aussi être utilisée pour générer un code RTL (*Register Transfer Level*) du processeur.

On retrouve également plusieurs projets académiques. StepNP [PAPB02] et Roses [CLNP02] sont deux environnements de développement utilisant une méthodologie de réutilisation de blocs de propriété intellectuelle. Alors que StepNP se concentre sur le développement de processeurs réseaux, l'emphase de Roses est plutôt mise sur l'accélération du raffinement des plates-formes, en offrant des mécanismes de génération automatique des adaptateurs matériels et des API (*Application Programming Interface*) logiciels. Teepee, développé par le groupe Mescal [MIHA02], est un autre bon exemple d'outil de développement.

Pour ce travail, différentes raisons ont motivé le choix de StepNP :

- une bibliothèque de blocs de propriété intellectuelle est disponible;

- les architectures en développement sont facilement modifiables par l'ajout ou le changement de composants;
- tous les outils essentiels au développement d'une architecture haut niveau (simulateur, débogueur et différents mécanismes d'analyse de performance) sont fournis et simples d'utilisation;
- et surtout, pour ce contexte académique, le code source est disponible et gratuit.

Objectifs

L'objectif principal de ce travail est d'évaluer, à haut niveau, les éventuels bénéfices de l'utilisation des processeurs configurables dans la conception des processeurs réseaux. Comme ces SoC possèdent généralement une grande panoplie de processeurs embarqués, souvent pourvus d'un jeu d'instructions spécialisées, les processeurs configurables semblent parfaitement adaptés à cet environnement. En effet, ils offrent plusieurs avantages, dont leur grande configurabilité, la possibilité d'ajouter des instructions et la génération automatisée des outils de développement logiciel. Ces atouts peuvent occasionner dans plusieurs cas des gains de performance intéressants ou encore une diminution importante du temps de développement.

Méthodologie

Afin de répondre aux objectifs fixés, plusieurs tâches devront être réalisées. Une première étape est l'exploration des opportunités offertes par la plate-forme StepNP. Comme elle sera utilisée pour la modélisation de processeurs réseaux, elle devra être bien comprise. De plus, comme elle n'offre actuellement aucun soutien pour l'utilisation de processeurs configurables, certains ajouts devront y être apportés. L'intégration du Xtensa de Tensilica [TENS03] a été retenue, principalement pour la simplicité de son processus de configuration et d'ajout d'instructions, ainsi que sa disponibilité.

L'exploration architecturale, basée sur l'utilisation des processeurs configurables, utilisera donc les mécanismes et les outils fournis par StepNP et ses nouvelles

possibilités. Deux applications réseaux complètes préalablement développées, soit IPv4 (*Internet Protocol version 4*) et IPsec (*Internet Protocol for security*), serviront de bancs d'essai. Les effets seront alors observés pour l'ensemble des tâches réseaux généralement réalisées, et non pas pour une seule fonction tel un algorithme d'encryptage ou le filtrage de paquets.

Originalité et contribution

Dans ce travail, plusieurs tâches spécifiques ont dû être réalisées. Toutefois, son originalité provient plutôt des modifications proposées à l'approche conventionnelle de co-design. Afin de bien tenir compte des nouvelles opportunités offertes par les processeurs configurables dans le processus de recherche architecturale, il a fallu se doter d'une méthodologie de recherche appropriée [QLBA04]. On propose donc d'ajouter certaines étapes à la méthodologie conventionnelle, principalement au niveau du partitionnement logiciel/matériel. Ces nouvelles étapes peuvent alors guider adéquatement les concepteurs de systèmes dans leurs décisions architecturales, à savoir s'il est avantageux d'ajouter des instructions spécialisées ou s'il est préférable de concevoir un coprocesseur.

Distribution des chapitres

Ce mémoire est constitué de quatre chapitres. Le chapitre 1 survole le monde des processeurs réseaux, incluant son environnement, ses applications, différentes architectures ainsi que StepNP, l'outil privilégié dans ce projet pour le développement des processeurs. Le chapitre 2 décrit d'abord l'approche conventionnelle employée en co-design, puis présente les modifications proposées pour l'intégration des processeurs configurables dans le processus de recherche architecturale. Le chapitre 3 présente la méthodologie et les outils utilisés en exposant deux exemples de réalisation. Enfin, les résultats et les limitations du projet sont présentés au chapitre 4.

CHAPITRE 1

Revue des processeurs réseaux

Un processeur réseau est un SoC multiprocesseur destiné aux applications réseaux. Un tel système inclut généralement : un ensemble de processeurs et coprocesseurs dédiés à certaines tâches, une structure de mémoire avancée, des interconnections hautes vitesses ainsi que des interfaces spécialisées pour un traitement de paquets efficace. Leur architecture est influencée par un grand nombre de facteurs, dont le débit visé, le type de traitement effectué et la flexibilité face aux nouveaux protocoles. On retrouve actuellement sur le marché une grande variété de processeurs réseaux, passant de solutions exclusivement matérielles aux architectures reposant sur des GPP (*General Purpose Processor*).

Le présent chapitre débute par une introduction à l'environnement des processeurs réseaux. Par la suite, une brève revue des applications en jeu est présentée puisque celles-ci jouent un rôle déterminant dans le processus de conception de ces SoC. Différentes architectures existantes sont ensuite présentées et certaines tendances sont dégagées. Finalement, StepNP, soit la plate-forme privilégiée dans ce travail pour la recherche architecturale, est présentée.

1.1. Environnement des processeurs réseaux : le routeur

Internet repose aujourd'hui sur chacun des ordinateurs branchés, mais encore plus sur l'infrastructure qui en permet l'interopérabilité. Le cœur de ce réseau de télécommunication est principalement composé d'équipements de transmission (généralement optiques) et de routeurs. Alors que les systèmes de transmission ne servent qu'au transport des données, les routeurs doivent traiter l'information qu'elles contiennent de façon à les acheminer convenablement dans le réseau. L'architecture des routeurs dépend d'une multitude de facteurs. Néanmoins, leur fondation est généralement la même. La Figure 1.1 présente le squelette typique d'un routeur à haute performance

[PMCS99], [CHAO02]. Un tel système est essentiellement composé d'un ensemble de N cartes de ligne (*line cards*) et d'un commutateur NxN.

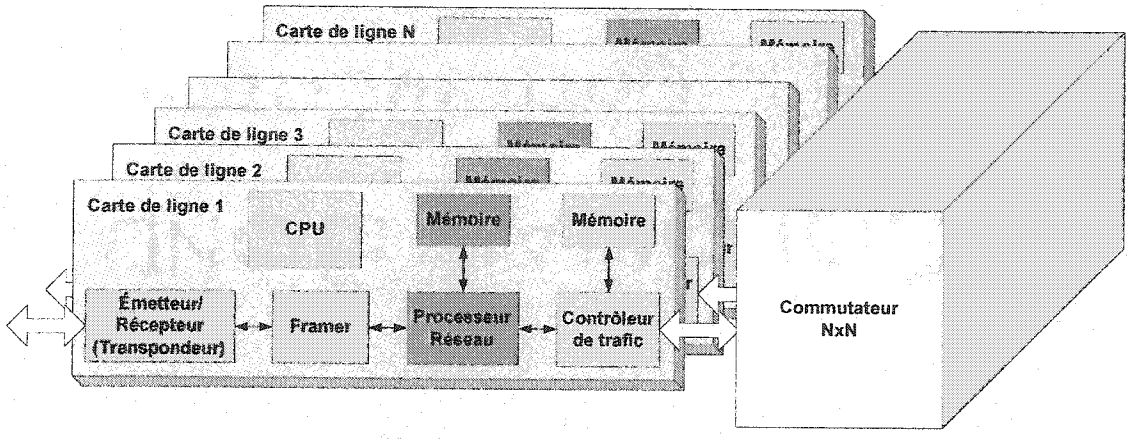

Figure 1.1 : Architecture typique d'un routeur à haute performance

Chaque carte est responsable d'une partie du traitement des paquets entrants et sortants d'une ou plusieurs interfaces (souvent nommé port). Le nombre de ports dépend de l'environnement pour lequel le routeur a été conçu. Par exemple, un routeur à utilisation domestique peut ne contenir que 4 ports Ethernet à 100 Mbps chacun. Pour les routeurs d'infrastructure, plusieurs interfaces OC-3 (*Optical Carrier 3*), OC-12, OC-48 et OC-192 doivent être supportées. Toujours pour ces systèmes, un commutateur performant (*swtich fabric*) est aussi nécessaire pour le transfert des paquets entre les différentes cartes de ligne.

Les cartes de ligne sont généralement composées de quelques éléments, dont les principaux sont un émetteur/récepteur, un *framer*, un contrôleur de trafic, un CPU, des mémoires spécialisées et finalement un processeur réseau.

- **Émetteur/Récepteur** Ce module est responsable de l'émission et de la réception des paquets. Comme l'information est souvent transmise par des signaux sériels optiques mais traitée sous forme électrique, ce module doit aussi réaliser la conversion de

signaux optiques à électriques et vis versa. Un procédé similaire s'applique aux systèmes sans fil.

- ***Framer*** : Le *framer* synchronise, délimite et modifie les paquets selon les requis du medium utilisé. Alors que l'émetteur/récepteur traite l'information sous formes de données continues, le *framer* doit reconnaître les structures utilisées et ainsi définir le début et la fin des trames.

- **Contrôleur de trafic :** Afin de rencontrer les spécifications de chaque service (priorité, débit maximal, tolérance aux pertes, etc.), le contrôleur de trafic accomplit différentes fonctions de contrôle sur les flots de données, incluant la gestion des mémoires tampons et l'ordonnancement des paquets, s'il y a lieu.

- **CPU :** Dans les systèmes à haute performance, un processeur d'usage général est souvent utilisé pour réaliser la logique de contrôle (voir section 1.2.4). Toutefois, pour des systèmes de moindre importance, la mise à jour des tables de routage et le traitement des cas d'exception peuvent être réalisés par le processeur réseau lui-même.

- **Processeur réseau :** Bien que tous les constituants des cartes de ligne soient essentiels, on s'accorde généralement pour dire que le processeur réseau en est le principal élément. C'est lui qui effectue à proprement parler le traitement des paquets. Ses principales tâches sont la consultation de tables, la classification et la modification des paquets. Dans les systèmes à haute performance, on retrouve souvent deux exemplaires du même processeur réseau par carte de ligne [MABA03] : un pour le traitement des paquets entrants, l'autre pour le traitement des paquets sortants.

1.2. Rappel sur les applications réseaux

Le but ici visé n'est pas d'approfondir certains protocoles, mais plutôt de dresser un portrait général des applications réseaux; et surtout de bien identifier les tâches qui doivent être réalisées.

1.2.1. Modèle OSI (*Open System Interconnection*)

Le modèle OSI [STAL99] a été développé en 1978 par l'organisation internationale de standardisation (ISO) de façon à fournir un modèle d'architecture pour les communications entre systèmes distants. La Figure 1.2 présente le modèle OSI comprenant sept couches, soit en partant du bas : la couche physique, la couche liaison de données, la couche réseau, la couche transport, la couche session, la couche présentation et la couche application. Quelques protocoles sont aussi donnés pour bien situer les différents niveaux. Chaque couche définit clairement le travail à réaliser et les relations avec ses couches adjacentes, permettant ainsi l'implémentation ou la modification d'une couche indépendamment des autres. Ce modèle se voit donc une solution aux problèmes d'interopérabilité des équipements hétérogènes actuels. Il est d'ailleurs utilisé comme référence pour la conception des protocoles réseaux.

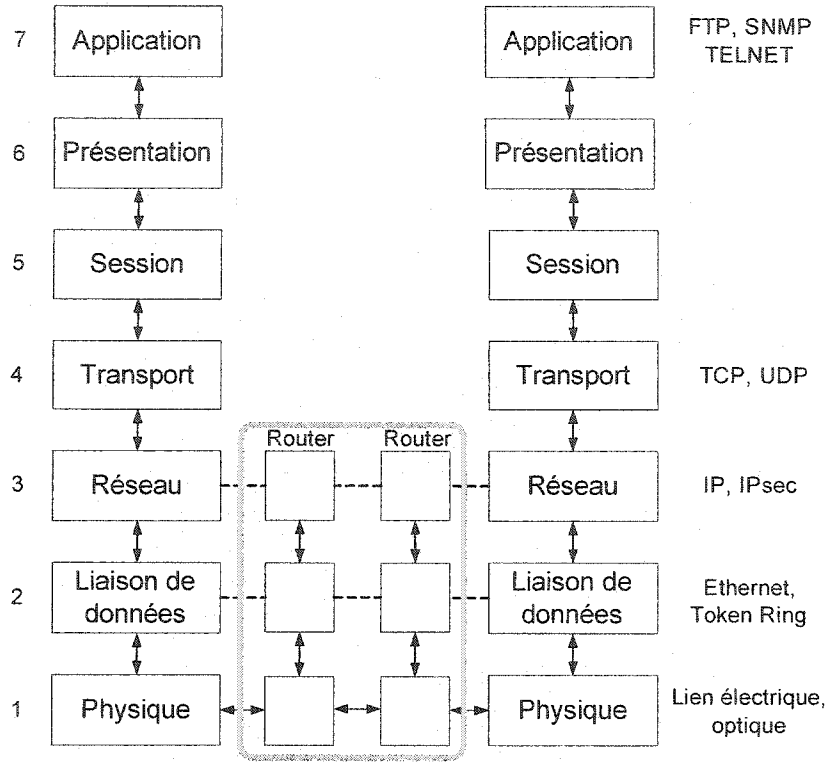

Figure 1.2 : Modèle en couches OSI

Les couches basses (1, 2 et 3) sont en charge de la transmission de l'information. Généralement réalisées par un ensemble de modules matériels et logiciels, elles définissent l'acheminement des données entre les systèmes distants. Alors qu'elles sont complètement implémentées par les stations hôtes, les appareils assurant la connectivité sur les réseaux n'implémentent parfois que certaines couches, dépendamment du niveau du traitement réalisé. Par exemple, les concentrateurs n'implémentent que la couche physique, alors que les routeurs effectue un traitement au niveau de l'ensemble des couches basses. Les couches hautes (5, 6 et 7) s'occupent du traitement de l'information échangée, indépendamment du moyen de connexion utilisé. Ces couches n'interviennent généralement qu'entre les stations hôtes. Finalement, la couche 4 gère quant à elle le transfert de l'information entre les couches basses et hautes. Une brève définition est ici donnée pour chacune de ces couches.

- **La couche physique :** La couche physique est responsable de la transmission des bits sur un canal de communication. Elle est entre autre concernée par les caractéristiques mécaniques, électriques et fonctionnelles du medium utilisé.

- **La couche liaison de données :** Cette couche doit fournir à la couche réseau une liaison fiable au travers le lien physique. Elle synchronise et contrôle le débit de la transmission des données sous formes de trames. Elle s'occupe parfois de la détection et de la correction d'éventuelles erreurs survenues à la couche physique.

- **La couche réseau :** Cette couche achemine les paquets de n'importe quel nœud du réseau vers n'importe quel autre. Elle doit entre autres gérer les problèmes de congestion dans le réseau et les différents mécanismes de routage.

- **Couche transport :** La fonction de base de la couche transport est de s'assurer de l'acheminement des messages au destinataire. Pour ce faire, elle doit traiter les données de la couche session et les découper au besoin en petites unités pour enfin les passer à la couche réseau. Le réassemblage et l'ordonnancement des messages sont effectués à la réception lorsque nécessaire. Cette couche est également responsable de l'établissement et du relâchement de connexions sur le réseau lorsque désiré.

- **La couche session :** Cette couche organise et synchronise les communications entre différentes tâches distantes. Elle permet aussi d'insérer des points de reprise dans le flot de données de manière à pouvoir reprendre le transfert en cas d'échec.

- **La couche présentation :** La couche présentation traite la syntaxe et à la sémantique des données transmises. Les opérations typiquement effectuées sur les données sont la conversion, le formatage, le cryptage et la compression.

- **La couche application :** Cette couche sert de point d'accès au réseau pour les programmes de l'utilisateur en lui fournissant certains services de base, tel le transfert de fichiers ou encore la messagerie.

Avant d'être transmise sur le réseau, l'information passe donc par chacune des sept étapes, de la couche application à la couche physique. Chaque étape ajoute généralement un en-tête supplémentaire aux données de la couche précédente afin d'offrir de nouveaux services. À l'arrivée, les données parcourent le chemin inverse, c'est-à-dire qu'elles circulent de la couche 1 à la couche 7 en traitant graduellement les en-têtes ajoutés.

1.2.2. Modèle TCP/IP (*Transmission Control Protocol/Internet Protocol*)

Le modèle OSI n'est toutefois qu'une référence. Celui le plus utilisé sur Internet, le modèle TCP/IP, n'a que cinq couches. Les fonctionnalités des couches sont toutefois globalement les mêmes, seule la couche application regroupe dorénavant les couches 5, 6 et 7 du modèle OSI. Comme son nom l'indique, ce modèle repose essentiellement sur une suite de deux protocoles, soit IP pour l'implémentation de la couche réseau et TCP pour la couche transport.

1.2.3. Quelques applications en bref

Dans le cadre de ce travail, nous considérons principalement les tâches réalisées aux couches 2 à 4 du modèle OSI, c'est-à-dire aux protocoles supportés par les routeurs et généralement implantés par les processeurs réseaux. Cette section en décrit quelques-uns parmi les plus répandus. Le lecteur est invité à lire [STAL99] pour une meilleure vue d'ensemble.

1.2.3.1. Ethernet (couches de niveau 1 et 2)

Développé au milieu des années 70, Ethernet [SPUR03] est le protocole actuellement le plus utilisé pour l'échange de données dans un réseau local. Il définit une méthode d'accès nommé CSMA/CD *(Carrier Sense Multiple Access with Collision Detection)*, dont les étapes sont les suivantes :

1. Lorsqu'une station veut transmettre une trame, elle doit d'abord s'assurer de la disponibilité du médium, sinon elle doit attendre.
2. La station peut alors émettre sa trame sur le médium.
3. Si une collision est détectée (c'est-à-dire plus d'une station ont tenté d'émettre simultanément), un signal de brouillage doit être émis par la station détectrice afin d'en informer les autres sur le réseau. Sinon, le message est simplement transmis en entier et le processus est terminé.
4. Après une collision, la station doit attendre une certaine période de temps aléatoire avant de recommencer les étapes depuis le début.

Les trames sont envoyées à toutes les stations du même domaine de collisions. Ces domaines sont divisées par des commutateurs (niveau 2) et/ou des routeurs (niveau3). Lorsqu'une station détecte qu'il s'agit d'une trame lui étant destinée (par le biais d'une adresse spécifiée dans l'en-tête du message), elle la copie. La Figure 1.3 présente les différents champs d'une trame Ethernet II, soit celle la plus utilisée. Les principaux sont les adresses de la source et de la destination, le champ Type pour l'identification du protocole supérieur et un champ pour la détection d'erreur (FCS).

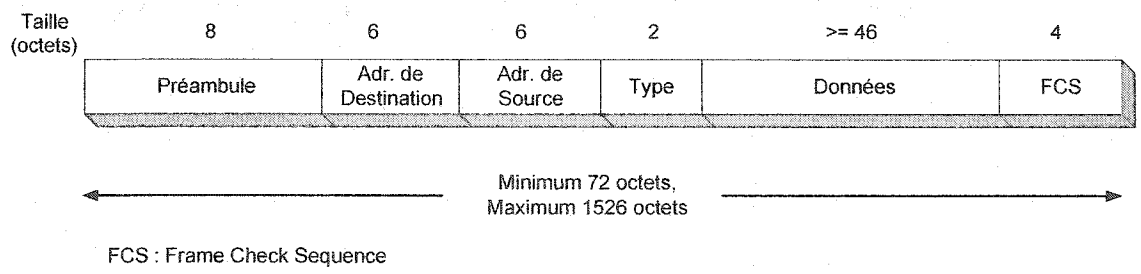

Figure 1.3 : En-tête d'un paquet Ethernet

Le protocole Ethernet définit aussi, en fonction des débits et des topologies voulus, différentes spécifications, tels le type de câblage, les longueurs minimales et maximales des câbles et des cordons, etc.

1.2.3.2. IPv4 (couche de niveau 3)

IPv4 est actuellement le protocole le plus répandu pour les communications de niveau 3. Il permet un acheminement efficace des paquets IP au travers un réseau grâce à l'utilisation d'adresses logiques ajoutées aux messages. La Figure 1.4 présente l'en-tête des paquets IPv4 alors que le Tableau 1.1 décrit brièvement chacun de ses champs.

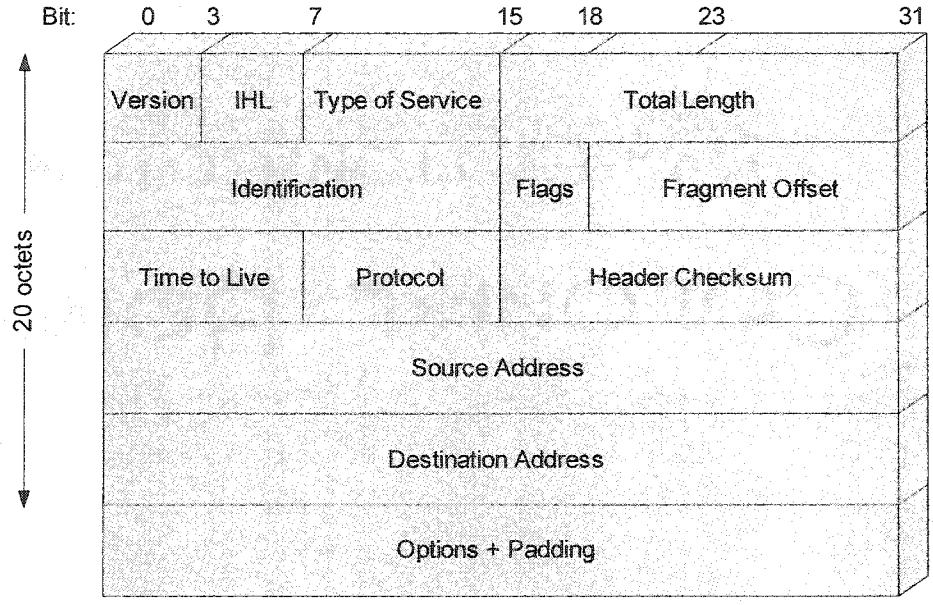

Figure 1.4 : En-tête d'un paquet IP

Tableau 1.1 : Champs de l'en-tête d'un paquet IP

Champs	Description
Version	Version du protocole IP utilisée
IHL	Taille de l'en-tête (en mots de 32 bits)
Type of Service	Indique le type de service applicable au datagramme
Total Length	Taille du paquet entier (en octets)
Identification	Numéro permettant d'identifier le paquet
Flags	Le premier bit est inutilisé Le second indique si le datagramme peut être fragmenté Le troisième indique si le datagramme est le dernier fragment
Fragment Offset	Indique où le fragment se retrouve dans le datagramme original
Time to Live	Nombre maximal de routeurs à travers lesquels le paquet peut passer.
Protocol	Indique le protocole de la couche supérieure utilisé
Header Checksum	Code permettant la détection d'erreurs de l'en-tête
Source Address	Adresse IP de la source
Destination Address	Adresse IP de la destination
Options + Padding	Permet l'ajout d'options et assure que l'en-tête est un multiple de 32 bits

Brièvement, les étapes à suivre pour le routage des paquets IP sont les suivantes.

1. Récupérer un paquet en entrée.
2. Vérifier la validité des différents champs.
3. Trouver la destination prochaine du paquet (cette étape implique la consultation d'une table de routage). Il existe différentes techniques pour le calcul de la destination et la gestion des tables de routage. Néanmoins, la détermination du chemin utilise généralement l'algorithme du *longest prefix match*, permettant une réduction significative de la taille des tables. Dans le cas où la destination est directement reliée au routeur, le protocole ARP (*Address Resolution Protocol*) est utilisé afin d'obtenir l'adresse réelle du destinataire à partir de son adresse logique.
4. Traiter différents champs : décrémentation du TTL (*Time to Live*), traitement des options, mise à jour du Header Checksum, etc.
5. Fragmenter le paquet si le réseau suivant exige des paquets de taille inférieure à celle actuelle. Cette tâche implique la vérification du drapeau de fragmentation, le découpage des données et l'ajustement des différents champs impliqués.

L'assemblage du paquet devra être réalisé lorsque qu'un traitement d'une couche supérieure sera nécessaire.

6. Insérer le paquet dans la bonne file d'attente en sortie.

IPv4 spécifie aussi le format des messages ICMP (*Internet Control Message Protocol*) devant être envoyés lorsque des erreurs ou des problèmes sont détectés, par exemple le champ TTL expiré ou encore s'il est impossible d'atteindre la destination.

Bien qu'actuellement largement utilisé, IPv4 tend à être remplacé par l'*Internet Protocol version 6* (IPv6). Très similaire au précédent protocole, IPv6 propose quelques modifications afin de surmonter certaines limitations. Les principaux changements sont les suivants.

- Les adresses sont codées sur 128 bits (au lieu de 32 bits).
- La fragmentation et l'assemblage des données ne sont plus supportés au niveau des routeurs afin d'alléger leurs tâches. L'émetteur doit s'assurer que les paquets envoyés sont de tailles supportées.
- Toujours dans le but de réduire le traitement effectué par les routeurs, le champ *Header Checksum* est retiré de l'en-tête IP. Les mécanismes de détection d'erreurs doivent être mis en oeuvre par les protocoles de niveau supérieur.
- Des champs ont été ajoutés à l'en-tête pour intégrer IPsec (défini plus bas) et des fonctions de qualité de service (QoS).

1.2.3.3. IPsec (couche de niveau 3)

IPsec [KEAT98a] fournit des services supplémentaires à la couche 3 du modèle OSI pour sécuriser les communications au travers les LAN (*Local Area Network*), les WAN (*Wide Area Network*) ou encore Internet. Il offre principalement des services d'authentification et de confidentialité (chiffrement) par le biais de deux protocoles, soit *IP Authentication Header* (AH) [KEAT98b] et *Encapsulating Security Payload* (ESP) [KEAT98c]. Ces protocoles supportent deux modes : le mode transport, dans lequel seules les données des protocoles des couches supérieures sont protégées, et le mode tunnel, dans lequel les services de sécurité sont appliqués aux paquets IP, eux-mêmes encapsulés dans de

nouveaux paquets IP. La Figure 1.5 présente ces deux modes où les sections ombragées correspondent aux données protégées.

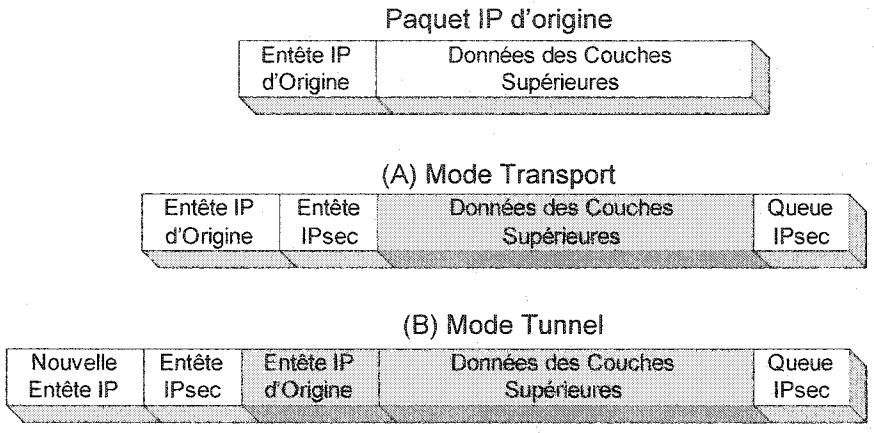

Figure 1.5 : Mode Tunnel ou Transport

Plus précisément, le **protocole AH** fournit les protections suivantes :

- intégrité des données sans connexion : assure qu'aucune modification du contenu d'un paquet en transition n'est possible sans être détectée;
- authentification de l'origine : permet de s'assurer de l'identité de la source;
- protection contre la reprise de paquets.

Le **protocole ESP** fournit quant à lui tous les services offerts par AH, plus :

- la confidentialité des données par cryptage. Plusieurs algorithmes peuvent ici être utilisés, dont les plus connus sont : DES (*Data Encryption Standard*), Triple DES, CAST-128, RC5, IDEA (*International Data Encryption Algorithm*) et Blowfish;
- le contrôle limité de la confidentialité des flux de données.

Bien sûr, ces protocoles ne seraient possibles sans un mécanisme d'échange de clés. IPsec supporte donc aussi différents algorithmes pour la gestion et l'échange des clés.

1.2.3.4. NAT (*Network Address Translation*, couches de niveau 3 et 4)

Un des problèmes au routage actuel des paquets sur l'Internet est l'assignation des adresses IP. Pour être accessibles par les routeurs dorsaux, tous les appareils connectés doivent avoir une adresse globalement unique. Comme il en existe un nombre limité d'adresses, les fournisseurs de services doivent alors se débrouiller avec un nombre restreint. De plus, les tarifs sont souvent proportionnels au nombre d'adresses possédées.

Le protocole NAT résout partiellement ce problème en permettant aux machines d'un réseau local d'être représentées par la même adresse. Lorsqu'un paquet doit sortir du réseau privé, NAT remplace son adresse interne par une adresse temporaire globale en prenant bien soin d'enregistrer la correspondance. Un lien entre la machine locale et le monde extérieur est alors établi pour toute la durée de la connexion. Lorsque l'application est terminée, l'adresse globale est alors libérée et de nouveau disponible pour une autre application (sur le même ordinateur ou n'importe quels autres sur le réseau). Il suffit donc de posséder une ou quelques adresses globalement valides et de la ou de les partager entre les ordinateurs locaux. NAT s'appuie sur le principe que toutes les machines ne requièrent pas d'accès à Internet simultanément.

NAT apportent plusieurs avantages :

- permet à un grand nombre d'usagers d'accéder à Internet en n'utilisant qu'une ou que quelques adresses globales;
- augmente le niveau de sécurité des réseaux privés en cachant leur structure;
- comme les adresses sont locales, il est facile de modifier les adresses IP globales sans avoir à reconfigurer les machines à l'interne.

1.2.3.5. Mécanismes de QoS (*Quality of Service*, couches 3 à 7)

Sans entrer dans les détails, la qualité de service réfère à la capacité d'un réseau d'offrir différents niveaux de services pour différents paquets, en utilisant des mécanismes de priorisation et/ou de réservation de bande passante. Les paquets peuvent être classés selon leur flot (adresses et ports de la source et de la destination) ou encore le type de

données qu'ils transportent (fichier, voix, vidéos, etc.). Les mécanismes de QoS concernent principalement la gestion des paquets et n'affectent pas le traitement même des donnés.

1.2.4. Tâches à effectuer

Bien qu'il existe une panoplie de protocoles implémentant les couches basses du modèle OSI, le traitement à effectuer d'une réalisation à l'autre est toujours très similaire et peut grossièrement être décomposé en 6 catégories, soit le filtrage, la consultation de tables, les calculs, la manipulation de données, la gestion des files d'attente et la logique de contrôle [SHAW01], [AGER99b]. Toutefois, dépendamment de l'application, certaines catégories sont plus prépondérantes que d'autres. Pour être performant, un processeur réseau se doit d'excelller dans l'ensemble de ces catégories par des choix judicieux au niveau de son architecture. Chacune des catégories est ici reprise plus en détail.

Filtrage (*pattern matching*)

Le filtrage consiste principalement à comparer différents champs d'une trame ou d'un paquet avec des expressions connues. Les champs sont de tailles variables (plus souvent de 4, 8, 16, 32 et 48 bits). Par exemple, pour déterminer le sous protocole d'un paquet transporté par une trame Ethernet, on doit d'abord isoler le champ Type (voir Figure 1.3) de l'en-tête Ethernet, puis le comparer à des valeurs connues (0x8000 pour IP ou encore 0x806 pour ARP).

Consultation de tables (*lookup*)

Cette catégorie consiste à chercher, dans une base de données, l'entrée associée à une clé. Les structures de données et les algorithmes utilisés son très variés et dépendent de la dimension de la base de données, du nombre et de la taille des clés. Plusieurs requêtes peuvent être effectuées à différentes tables pour un même paquet. Par exemple, il est possible d'effectuer une recherche dans une table de correspondance pour savoir s'il existe déjà un flot associé à un paquet IP (pour une application de NAT), suivie d'une

seconde requête cette fois à une table de routage pour en déterminer la prochaine destination.

Calculs

Le routage des paquets nécessite une grande variété de calculs. On y retrouve, par exemple pour les protocoles Ethernet et IPv4, les calculs du CRC (*Cyclic Redundancy Check*) et du Header Checksum. IPsec y apporte aussi un lot important en ajoutant plusieurs algorithmes de chiffrement et d'authentification, très lourds en boucles de contrôle et en opérations mathématiques.

Manipulation de données

Cette catégorie inclut toutes les opérations affectant l'en-tête d'un paquet. On y retrouve la modification de certains champs (par exemple, l'ajustement de la prochaine destination d'un paquet IP) ou encore l'ajout de champs ou d'en-têtes (par exemple, pour l'intégration d'IPsec).

Gestion des files d'attente

La gestion des files d'attente consiste à contrôler la circulation des paquets provenant des interfaces d'entrée jusqu'aux bonnes interfaces de sortie, tout en coordonnant leurs déplacements avec les différents éléments qui doivent l'accéder. Elle doit tenir compte de plusieurs contraintes, entre autre pour la régulation du trafic, la segmentation et l'assemblage des paquets ainsi que l'application des mécanismes de QoS.

Logique de contrôle

Cette catégorie inclut généralement toutes les tâches qui n'affectent pas directement les paquets et qui ne requièrent pas de standards élevés de performance. On y retrouve principalement le traitement d'exceptions, la mise à jour des tables de routage et la collecte de statistiques pour fin d'analyse ou de facturation.

1.3. Architecture des processeurs réseaux

Les processeurs réseaux commerciaux sont très variés et comprennent tous les styles d'architectures. Malgré cette grande diversité dans les approches, on peut tout de même remarquer certaines tendances actuelles et prévoir les directions futures.

1.3.1. Quelques processeurs réseaux commerciaux

Nous ne présentons dans ce mémoire que certains processeurs réseaux actuellement disponibles. Toutefois, [SHAH01] fournit une excellente analyse du marché.

1.3.1.1. IBM (PowerNP)

Le NP4GS3 de la famille PowerNP d'IBM [IBM03] est un processeur réseau à haute performance conçu pour le traitement de paquets allant jusqu'à 2.5 Gbps. Caractérisé par un processeur central PowerPC et 16 processeurs de protocoles, le NP4GS3 est principalement destiné au traitement rapide des couches 2 à 4 du modèle OSI.

Architecture

Bien que le NP4GS3 intègre plusieurs dispositifs dont un engin de recherche et du support pour les mécanismes de QoS, le cœur de ce processeur réseau réside en ce qu'IBM appelle le processeur embarqué complexe. Composé d'un PowerPC 405 spécialisé allant à 133MHz et de 16 processeurs de protocoles programmables, le processeur embarqué complexe est responsable du traitement des paquets entrants et sortants.

Chacun des 16 processeurs de protocoles possède 2 processus, pour un total de 32, dont 16 s'exécutent simultanément. Pour cacher la latence, un dispositif permet de permuter le processus s'exécutant lorsque qu'il est arrêté sur un accès bloquant. Comme chaque processus possède une copie de tous les registres incluant les registres d'états, ce changement de contexte est réalisé sans pénalité, c'est-à-dire en 0 cycle d'horloge [EELS97], [VITK00]. De plus, les processeurs de protocoles sont regroupés par deux, formant 8 *Dyadic Protocol Processors Units* (DPPUs). Chaque DPPUs se partagent un

ensemble de 9 coprocesseurs matériels dédiés, dont un module pour le calcul et la vérification du Header Checksum, un coprocesseur de recherche en arbre, un contrôleur de sémaphore, un compteur et différents modules accélérant les accès au bus. Toutefois, ils doivent être programmés en assembleur avec l'assistance d'outils de développement logiciel.

Quant à lui, le PowerPC possède des caches d'instructions et de données de 16 Ko. Il peut entre autre être affecté aux tâches de contrôle et de gestion si désiré.

1.3.1.2. Intel (IXP2800)

Basé sur le IXP1200, le IXP2800 d'Intel [INTE03] fait partie de leur deuxième génération de processeurs réseaux pouvant soutenir un débit équivalent au OC-192, soit de 10 Gbps. En plus d'offrir un traitement parallèle de haute performance, la seconde génération intègre la technologie *Hyper Task Chaining* d'Intel. Cette approche permet la décomposition du traitement d'un paquet en plusieurs étapes séquentielles. Afin de traiter les opérations interdépendantes efficacement, l'architecture est élaborée de façon à offrir des moyens rapides et flexibles au partage des données et aux communications entre différents processus et microprocesseurs.

Architecture

Les principaux constituants sont les suivants.

- 16 RISC (*Reduced Instruction Set Computer*) multiprocessus (*multithreaded*) de 1 ou 1,4 GHz possédants une mémoire d'instructions de 8 Ko pour le traitement des paquets.
- Un processeur embarqué de 32 bits fonctionnant à 700 MHz pour la logique de contrôle, incluant la gestion des tables de routage et l'exécution d'algorithmes complexes. Ce RISC possède des caches d'instructions et de données de 32 Ko.
- Plusieurs interfaces, dont 4 QDR SRAM pour les tables de routages performantes et une interface PCI standard pour l'ajout de matériels externes.

- De la mémoire de travail de 16 Ko et différents modules matériels, dont un coprocesseur pour le support d'IPsec, un engin de hachage, un UART sériel et des compteurs.

La plate-forme IPX2800 offre aussi des outils de développement et de simulation matériel et logiciel pour une mise en place rapide d'applications réseaux.

1.3.1.3. Motorola (C-5e)

Le C-5e de Motorola [MOTO03] est un processeur réseau destiné aux applications des couches 2 à 7. Il permet le traitement des paquets à 2,5 Gbps.

Architecture

Ce processeur réseau inclut un processeur d'usage général pour des fins de contrôle, 16 processeurs spécialisés ainsi que 5 coprocesseurs matériels partagés pour le traitement des paquets.

Les processeurs spécialisés sont composés d'un RISC standard et deux processeurs de données séries (SDP), l'un pour la réception, l'autre pour la transmission des paquets. Ils peuvent être soit programmés pour le support de paquets sous SONET (*Synchronous Optical NETwork*), Ethernet ou encore ATM (*Asynchronous Transfer Mode*). À la réception, le SDP attitré effectue une conversion série/parallèle des paquets, interprète les données et extrait les en-têtes, puis initie différentes requêtes aux tables de routage. Pour la transmission, l'autre SDP réorganise les en-têtes et les données des paquets, puis les convertit en données séries.

De son côté, le processeur RISC doit d'abord collecter les résultats des différentes requêtes effectuées par le SDP de réception et filtrer les données. En fonctions des résultats recueillis, il doit alors prendre des décisions sur l'acheminement et l'ordonnancement des paquets. Au besoin, d'autres requêtes peuvent être commandées. Dans leurs tâches, les processeurs RISC sont aidés par 5 coprocesseurs partagés, soit un

processeur responsable de la coordination avec d'éventuels processeurs externes, un module en charge de la coordination avec le commutateur pour l'utilisation de plusieurs C-5e, un gestionnaire des tables de routage, un contrôleur d'accès mémoire et un gestionnaire de queues de paquets.

De plus, le 5-Ce contient 3 bus internes indépendants (un bus de données global pour le transfert des paquets et de leurs descripteurs, un bus en anneau pour les communications inter processeurs et un bus global pour les communications inter processeurs via de la mémoire) offrant une largeur de bande cumulée de 76,6 Gbps. Il est programmable en langage C et C++.

1.3.2. Tendances actuelles et directions futures

Les processeurs réseaux performants se doivent d'être d'abord et avant tout en mesure de traiter des flots de paquets à des débits très élevés. Ils se doivent par la suite d'être flexibles pour supporter les nouvelles technologies et extensibles pour rencontrer la demande croissante en largeur de bande. Pour l'atteinte de ces objectifs, plusieurs défis de taille attendent les concepteurs architecturaux. En plus des contraintes temporelles et économiques, on compte la complexité et la diversité croissantes des applications réseaux, la largeur de bande restreinte disponible sur une même puce, la latence élevée des accès mémoire extérieurs et la mise à l'échelle du produit.

Dans leur conception, les designers ne sont limités que par leur imagination et peuvent donc faire preuve d'une grande créativité. Les sous-sections suivantes résument les principales approches employées.

1.3.2.1. Architectures multiprocesseurs

De façon à exploiter le parallélisme inhérent au traitement de paquets, l'ensemble des processeurs réseaux à haute performance semblent d'abord basés sur des architectures multiprocesseurs sur puce. Ce parallélisme découle de l'indépendance des données provenant de flots différents. Le principe est relativement simple et consiste

essentiellement à dupliquer le matériel de façon à traiter simultanément plusieurs flots de données, augmentant ainsi le débit total du traitement. Curieusement, les trois processeurs réseaux présentés possèdent chacun 16 processeurs spécialisés. Toutefois, pour l'ensemble des produits actuellement sur le marché, ce nombre est très variable. Certaines solutions ne possèdent qu'un seul processeur alors que le NP-1 de EZchip [EZCH03] en possède jusqu'à 64. De plus, un processeur d'usage général est souvent ajouté pour les tâches de gestion et de contrôle.

Avec un niveau d'intégration des SoC sans cesse croissant, on envisage un plus grand nombre de processeurs embarqués pour les futures générations de processeurs réseaux.

1.3.2.2. Utilisation de coprocesseurs matériels

Même si les processeurs embarqués sont souvent munis d'un jeu d'instructions adaptées aux tâches réseaux (voir section 1.2.4), l'ensemble du traitement n'est pas efficacement réalisé en logiciel. Comme chaque paquet doit être traité très rapidement, tous les processeurs réseaux ont recours à des coprocesseurs spécialisés. Un bon exemple est le matériel utilisé pour accélérer les accès au bus souvent très coûteux, dont le déplacement des paquets. Alors que plusieurs proposent d'accélérer cette tâche par l'ajout de coprocesseurs (entre autres le PowerNP et le IXP2800), Motorola propose plutôt avec leur C-5e d'inclure la logique nécessaire (SDP) à même les processeurs spécialisés. Voici d'autres exemples d'utilisation de coprocesseurs parmi les plus répandus.

- Gestionnaire des tables de routage : surtout utilisé pour accélérer la recherche dans une base de données.
- Gestionnaire de queues de paquets et support pour les fonctions de QoS.
- Modules pour le calcul de certains champs, dont le Checksum et le CRC.
- Coprocesseurs d'authentification et de chiffrement.

On inclut aussi dans cette section toutes les unités fonctionnelles spécialisées dans le traitement de paquets. Par exemple, il est commun d'ajouter aux processeurs des opérations accélérant le filtrage et la manipulation de bits.

1.3.2.3. Processeurs à plusieurs processus élémentaires

Toujours dans le but de cacher la latence de différents accès (par exemple, occasionnée lors du déplacement d'un paquet ou encore d'une requête à une table de routage), une autre solution (souvent utilisée conjointement avec l'ajout d'un coprocesseur) est d'utiliser des processeurs à plusieurs processus. Lorsqu'un processus doit attendre le résultat d'une action, un autre peut prendre sa place de façon à toujours garder le processeur occupé. Déjà grandement employé dans les systèmes conventionnels, la nouveauté réside ici dans l'intégration de dispositifs de changement de contexte rapide (par exemple, comme pour le NP4GS3 d'Intel), n'entraînant aucune pénalité.

1.3.2.4. Interconnexions sur puce

Avec un débit de traitement requis très élevé et un nombre de composants toujours croissants, les architectures centrées sur un seul bus ne procurent pas la largeur de bande nécessaire aux processeurs réseaux. On a recours à d'autres stratégies, dont des systèmes de communications sur puce propriétaires (par exemple, le bus ClearConnect de ClearSpeed [CLEA01a]) ou encore l'utilisation de plusieurs bus (par exemple, comme pour le 5-Ce).

Avec l'intégration d'applications de QoS, on devrait voir plus d'architectures de bus supportant plusieurs services. Un exemple actuel est le MSP5000 de BRECIS communication [BREC03], où des priorités d'accès au bus sont fixées en fonction du type des paquets traités.

1.3.2.5. Structure de mémoire avancée

Les architectures actuelles utilisent généralement un patron de mémoires complexe, où plusieurs mémoires de différents types et de tailles variables se retrouvent sur une même puce.

1.3.2.6. Outils de développement

La grande majorité des plates-formes proposent des outils avancés de conception logicielle et matérielle. Toutefois, bien qu'un compilateur C soit souvent fourni, les sections critiques doivent encore être réalisées en code assembleur pour bien prendre avantage des particularités d'une architecture.

1.4. StepNP : plate-forme de conception au niveau système

StepNP (*System-Level Exploration Platform for Network Processing*) est, comme son nom l'indique, utilisé comme plate-forme d'exploration architecturale de haut niveau pour processeurs réseaux. Il est essentiellement formé de trois éléments, soit d'un modèle d'architecture mono ou multiprocesseur, d'une ou plusieurs applications réseaux et d'outils de contrôle, de débogage et d'analyse de performance. La Figure 1.6 présente une application simple sur la plate-forme StepNP. Ces différents concepts sont repris plus en détails aux sections suivantes.

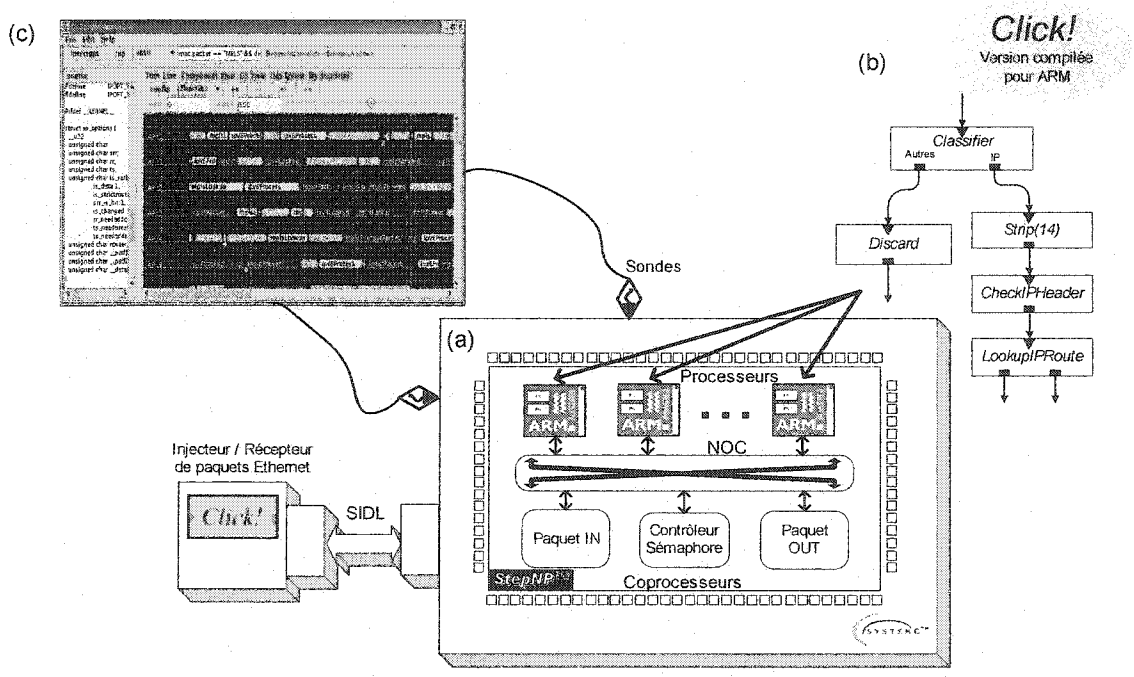

Figure 1.6 : Trois éléments d'une plate-forme StepNP : (a) une architecture, (b) une application réseau, (c) des outils de contrôle et d'analyse

1.4.1. Architecture (bibliothèque de composants réseaux)

À la base, StepNP possède une bibliothèque de modules SystemC 2.0 simulables représentant différents niveaux d'abstraction d'éléments réseaux. On en retrouve trois catégories, soit les processeurs, les modèles de communications et les accélérateurs matériels. Une architecture (Figure 1.6a) est simplement conçue en sélectionnant des éléments de différentes classes et en spécifiant leur agencement. Comme tous les modules implémentent la même interface de communication, il est alors facile de les connecter entre eux. De plus, toujours grâce à cette uniformité des interfaces, il est aussi possible de remplacer un élément sans avoir à modifier le reste de la plate-forme, accélérant ainsi l'exploration d'alternatives.

Processeurs

Actuellement, la bibliothèque de StepNP inclut quelques modèles d'ISS (*Instruction Set Simulator*) de différents RISC, dont celui du ARM v.4 et du PowerPC (versions 603, 603a et 604), tous deux du domaine public [GNU03]. Ces modèles fonctionnels *cycle accurate* sont aussi disponibles dans une version plus élaborée supportant le concept de multiprocessus matériels.

Modèles d'interconnexions

Le processus d'exploration architecturale débute généralement avec un système d'interconnexions simple. Dans StepNP, un canal de communication transactionnel est initialement employé. Un tel modèle simplifie l'interrelation des modules tout en accélérant les temps de simulation. La fonctionnalité de différentes architectures peut alors rapidement être vérifiée et les performances estimées.

Lorsqu'une famille d'architectures a été ciblée, un modèle d'interconnections répondant mieux aux exigences et plus fidèle à la réalité peut être choisi (ou encore développé). Différents prototypes de NoC (*Network-On-a-Chip*) sont déjà disponibles, dont des versions de Mesh 2D et de Cross Bar.

Coprocesseurs

Les processeurs réseaux commerciaux intègrent une grande variété d'accélérateurs matériels répondants généralement aux exigences propres de leur architecture. StepNP possède quelques coprocesseurs génériques, dont un contrôleur de sémaphores et des interfaces spécialisées. Toutefois, la recherche architecturale implique souvent l'exploration de nouveaux coprocesseurs dédiés. Basé sur SystemC 2.0, StepNP est alors un outil par excellence pour ce type de tâches, puisque la conception et l'intégration de nouveaux modules peuvent être réalisées très rapidement. De plus, une méthodologie de raffinement des modules est aussi intégrée à même SystemC.

1.4.2. Plate-forme de développement logiciel (Click)

À l'étape de la recherche architecturale, plusieurs solutions doivent être évaluées et confrontées entre elles. Pour ce faire, les architectures étudiées sont profilées en exécutant quelques applications réseaux pour lesquelles elles sont destinées. Comme une courte période de temps y est souvent allouée, on ne cherchera pas à établir précisément les performances finales, mais plutôt des estimés suffisamment précis pour discriminer les différentes architectures. Le développement d'un code optimal est donc écarté, d'autant plus qu'une application dédiée à une architecture spécialisée sera mise aux oubliettes si la solution n'est pas retenue.

En conséquence, on doit se tourner vers une plate-forme de développement rapide d'applications réseaux, dont le code est flexible et modulaire. De plus, comme les applications développées seront possiblement exécutées sur plusieurs processeurs différents, le code utilisé se doit d'être portable et de haut niveau. Pour ces raisons, le choix de StepNP s'est d'abord arrêté sur Click [KOHL01]. [YIPE02] fournit une étude comparative de trois différentes alternatives, dont Click, Princeton's Scout-based Extensible Router et Washington University's Router Plugins.

Click est un ensemble d'éléments réseaux originalement développé par le MIT. Ces éléments logiciels (programmés en C++) correspondent à tous les aspects du comportement d'un routeur, s'occupant entre autre du filtrage et de la modification des paquets, des communications avec les interfaces et de la logique de contrôle. Une application Click est simplement un regroupement de ces éléments interconnectés les uns aux autres de façon à produire le comportement souhaité. Pour ce faire, l'application est d'abord décrite dans un fichier de configuration en utilisant un langage propriétaire élémentaire. À l'exécution, le fichier est interprété par un analyseur syntaxique pour finalement produire le comportement désiré. C'est cette méthodologie qui rend Click très souple.

La Figure 1.6b présente un exemple où un premier élément '*Classifier*' rejette tous les paquets, excepté les paquets IP. '*Strip(14)*' enlève par la suite l'en-tête Ethernet de 14 octets des paquets restants, puis les envoie à l'élément '*CheckIPHeader*', responsable de la vérification de leur en-tête IP. Finalement, '*LookIPRoute*' effectue des requêtes pour le routage. En résumé, les paquets circulent d'un élément à l'autre, jusqu'à ce que leur traitement soit complet. Des applications plus complexes comme NAT ou encore IPv4 peuvent être développées de la même façon. Le Tableau 1.2 présente quelques-uns des éléments disponibles (il en existe environ 300) ainsi qu'une brève description pour chacun d'eux. Au besoin, il est aussi possible d'ajouter des éléments de son cru en respectant une syntaxe bien définie. Une version est aussi disponible pour des architectures multiprocesseurs [CHMO01].

Tableau 1.2 : Quelques éléments de Click

Éléments	Fonctionnalité
CkeckCRC32	Vérifie le champ CRC32
IPClassifier	Permet la classification de paquets IP (TCP, UDP, etc.)
IPFilter	Permet le filtrage de paquets IP
IPRewriter	Permet la traduction d'adresse IP (par exemple NAT)
IPsecAuthSHA1	Gère le protocole d'authentification SHA1
Queue	Gestion d'une queue de paquets
Shaper	Permet de limiter le débit d'un flot de paquets

Dans StepNP, il est aussi possible d'exécuter n'importe quelles autres applications (autre que Click), pour autant qu'elles puissent être compilées sur le processeur ciblé. Dans certains cas, un petit programme peut être très utile si on désir valider un algorithme ou encore s'assurer du bon fonctionnent d'un module matériel nouvellement conçu.

1.4.3. Outils de contrôle, d'analyse et de débogage

Un premier outil permet de contrôler et de visualiser sous différentes perspectives (architecturales, temporelles ou encore logicielles) l'exécution d'un modèle (la Figure 1.6c présente la vision temporelle). Il a comme objectif de fournir au concepteur une vue d'ensemble de l'architecture réalisée. Par la suite, on retrouve tous les outils spécifiques aux différents processeurs utilisés (compilateur, éditeur de liens, débogueur, profileur), indispensables au bon fonctionnement de certains modules.

Un outil supplémentaire est actuellement en développement [DESL04]. Il aura comme tâche de fournir une analyse plus poussée des transactions effectuées entre les différents éléments d'une plate-forme (taille des accès, débit, latence, etc.). Ces mesures aideront par la suite au développement et au raffinement de NoC plus appropriés.

CHAPITRE 2
Flot de conception

2.1. Flot de co-design actuel

Plusieurs voies distinctes sont empruntées par les ingénieurs dans la conception des SoC, autant dans les approches privilégiées que les outils utilisés. Néanmoins, les différentes étapes à réaliser demeurent grossièrement les mêmes, peu importe le chemin suivi. La Figure 2.1 présente l'approche générale (représentée par les lignes pleines).

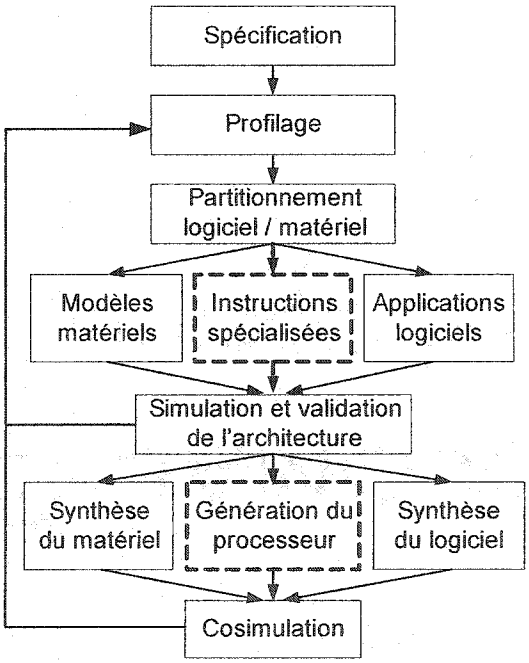

Figure 2.1 : Flot de co-design actuel (lignes pleines) et proposé (lignes pleines et pointillées)

Ce flot de conception usuel débute par une spécification du produit à concevoir. On y décrit toutes les contraintes, autant fonctionnelles que temporelles. Comme il est important de bien préciser et de vérifier les différentes exigences du design en début de

projet, une version entièrement logicielle de l'application est généralement développée à cette étape. Différents langages de haut niveau (par exemple, C++, SystemC, etc.) peuvent être employés pour le développement d'un modèle fonctionnel sans notion de temps [CBRB04].

L'étape suivante consiste à profiler l'application initiale et ses versions subséquentes, c'est-à-dire mesurer et analyser l'exécution de celles-ci sous différentes conditions. L'objectif est de bien identifier l'ensemble des tâches à réaliser tout en déterminant celles qui demandent plus de cycles d'exécution et plus de ressources.

Par la suite, une fois la fonctionnalité vérifiée, on doit s'assurer que les contraintes d'ordre temporelles sont aussi rencontrées. En cas de non-respect, les observations précédentes seront très utiles puisqu'elles serviront à identifier quelles tâches sont en cause. Comme le logiciel a plusieurs avantages sur le matériel, dont sa flexibilité et sa rapidité de développement, on va généralement tenter, en premier lieu, d'optimiser certaines tâches en retravaillant les algorithmes implantés. Toutefois, dans le cas où le logiciel ne donnerait pas les résultats escomptés, on devra accélérer les sections de code critiques par l'ajout de modules matériels. Les algorithmes réalisant des manipulations sur des bits ou utilisant des registres de dimensions peu communes sont souvent de bons candidats puisqu'ils peuvent être aisément accélérés en matériel. Dans d'autres cas, la parallélisation du traitement est souvent une solution envisagée.

Pour réduire significativement le temps de développement, l'utilisation de blocs de propriété intellectuelle existants est d'abord privilégiée, sinon, de nouveaux modules matériels seront développés. Cette phase se nomme le partitionnement logiciel/matériel, car l'on y décide quelles parties seront implantées en matériel et quelles autres le seront en logiciel. Comme plusieurs combinaisons possibles sont tentées, plusieurs simulations sont souvent nécessaires.

Les étapes du profilage, du partitionnement, ainsi que les différentes simulations, forment un processus itératif appelé la recherche architecturale de haut niveau. Ce processus, visant à déterminer les différents modules logiciels et matériels ainsi que leur agencement, est répété jusqu'à l'obtention des résultats requis. Le raffinement des modèles matériels est graduellement accompli et les notions du temps sont ajoutées d'une itération à l'autre. Pour qu'un environnement de recherche architecturale soit efficace, il doit être d'un niveau d'abstraction élevé afin de faciliter et d'accélérer l'exploration d'un grand nombre de solutions. Cependant, il se doit aussi d'être suffisamment détaillé et d'inclure la notion du temps pour fournir de bons estimés de la performance.

Les étapes subséquentes consistent, d'une part, à valider les hypothèses faites sur les modèles matériels précédemment développés. En effet, les délais spécifiés lors du processus de raffinement ne garantissent pas une correspondance avec la réalité. D'autre part, les interfaces des modules matériels et logiciels doivent être générées [CLNP02] et leur interopérabilité vérifiée. Pour ce faire, la synthèse du matériel et du logiciel est d'abord réalisée. Par la suite, une cosimulation complète est nécessaire. Comme ce travail se préoccupe de la recherche architecturale, ces dernières étapes ne seront pas abordées dans ce document.

Bien que ce flot de conception soit nécessaire au développement de tout nouveau design, il n'est pas souvent employé dans son entier par l'industrie. En effet, on préfère généralement commencer le processus avec une plate-forme existante, à laquelle on apporte certaines modifications afin de rencontrer les nouvelles caractéristiques exigées.

2.2. ASIP, ADSP, processeurs configurables et reconfigurables

Plusieurs possibilités s'offrent aux designers système pour l'implémentation des fonctionnalités d'une spécification dans les SoC. L'exécution de l'ensemble d'une application sur un ou plusieurs processeurs est une première alternative très attrayante, principalement dû à la rapidité de développement du logiciel et sa grande flexibilité. Des processeurs embarqués provenant de familles d'architectures bien connues (par exemple

ARM, MIPS) sont souvent utilisés. De tels processeurs possèdent toutefois un jeu d'instructions et une architecture généralement fixes et ne peuvent donc être optimisés pour une application spécifique. Comme plusieurs requièrent des standards de performance très contraignants (ce qui est particulièrement le cas pour les applications réseaux), l'accélération de certaines tâches est parfois nécessaire. Pour ces applications, plusieurs options supplémentaires sont envisageables, dont l'usage de processeurs spécifiques à un domaine d'applications (ADSP) ou encore l'ajout de matériel.

Bien que la performance des processeurs d'usage général ne cesse d'augmenter, la limite de leur efficacité et leur grande consommation de puissance les rendent peu compétitifs pour l'exécution de certaines applications embarquées. Les ADSP, dont les ASIP (*Application-Specific Instruction-set Processors*), concèdent un peu de flexibilité au profit d'une augmentation de performance. Comme les processeurs embarqués n'effectuent couramment qu'un nombre restreint de tâches, les ADSP offrent du support supplémentaire pour un ensemble de celles-ci. Ils possèdent quelques unités fonctionnelles spécialisées réalisant les patrons d'opérations les plus fréquemment rencontrées par le type d'applications visées. Par exemple, pour accélérer l'exécution d'un filtre FIR (*Finite Impulse Response*), une instruction réalisant la multiplication de deux nombres suivie d'une addition est souvent ajoutée. Des outils de développement logiciel adaptés sont aussi fournis [ARCO01].

Toutefois, les unités fonctionnelles ajoutées sont généralement simples et ne peuvent réaliser d'opérations complexes. Bien qu'intéressants, les gains rencontrés demeurent souvent mineurs. Dans certains cas critiques, l'ajout de coprocesseurs est donc essentiel. Ceux-ci peuvent améliorer de beaucoup les performances globales d'un système en dégageant les processeurs fixes des tâches complexes. En effet, les coprocesseurs matériels peuvent grandement accélérer certaines fonctions grâce à l'usage du traitement parallèle ou encore la réalisation de plusieurs instructions consécutives en un même

cycle. Ils sont cependant très coûteux en temps de développement et tendent à rendre les architectures inefficaces au changement.

Une autre alternative (encore au stade de la recherche) consiste à utiliser des processeurs possédant cette fois un coprocesseur reconfigurable (par exemple, les FPGA intégrant un processeur). Une telle approche permet la programmation dynamique de logique programmable pour l'exécution de différentes fonctions consécutives. La fonctionnalité du coprocesseur est alors modifiée en fonction des besoins présents et futurs du processeur. Bien que cette approche gagne en flexibilité sur les coprocesseurs fixes, la performance atteignable demeure limitée par la vitesse d'opération moindre de la logique programmable. La Figure 2.2 présente le compromis entre la performance et la flexibilité de différentes implémentations disponibles pour la conception des SoC [DUCH03].

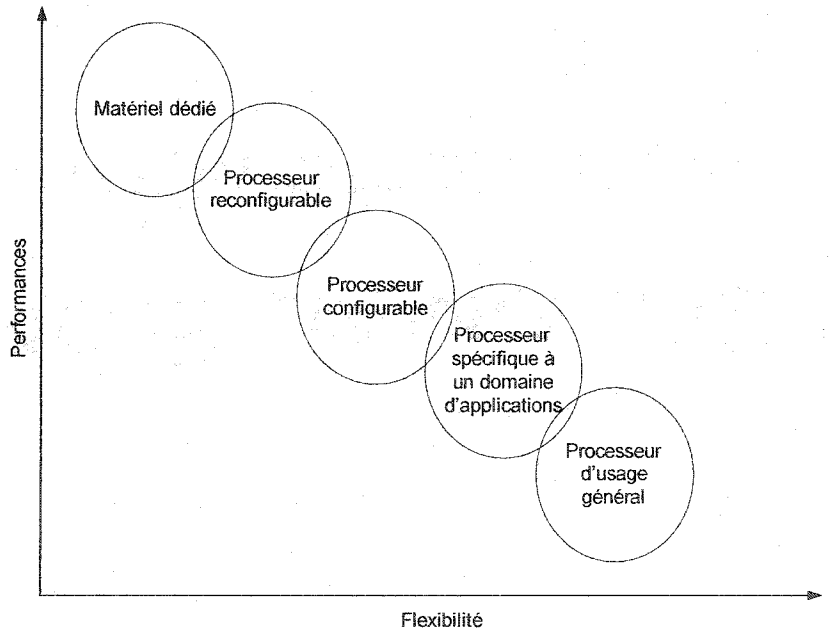

Figure 2.2 : Spectre des différentes technologies disponibles pour la conception des SoC illustrant leurs compromis au niveau de la performance et de la flexibilité

Au milieu de ce spectre se trouve une solution dont on n'a pas encore discuté, soit les processeurs configurables. Leur compromis entre la performance et la flexibilité fait d'eux une alternative très intéressante dans la conception des SoC. Cette technologie offre aux designers de systèmes un grand choix de processeurs. Non seulement le jeu d'instructions peut être optimisé pour une application, mais plusieurs options architecturales comme la profondeur du pipeline, le nombre et la dimension des bus ainsi que les caches et les interfaces mémoires peuvent aussi être ajustés.

Bien que la personnalisation d'un processeur semble des plus attrayante, elle apporte un lot de défis importants. Dans un premier temps, pour bien tirer avantage d'une nouvelle architecture, des outils adaptés sont nécessaires (compilateur, débogueur, simulateur, etc.). Toutefois, le temps de développement du logiciel ne doit pas pour autant être prolongé et d'ainsi pénaliser le potentiel d'efficacité de cette solution. D'autre part, comme de nouvelles architectures sont conçues, les étapes de la synthèse, de la validation et de la vérification deviennent des plus importantes.

Heureusement, ce qui rend encore plus intéressant l'usage des processeurs configurables, c'est que l'ensemble des compagnies (dont ARC, Improv et Tensilica) offrent déjà des mécanismes d'automatisation de ces tâches. Ils permettent ainsi le développement rapide de processeurs embarqués flexibles et performants.

2.2.1. Le Xtensa de Tensilica

À la base, le processeur configurable Xtensa est un processeur similaire au DLX à 5 étages [HEPA90]. Une instruction provenant du cache d'instructions est d'abord chargée au premier étage (*Load Instruction*). Comme les instructions sont codées sur 16, 24 ou 32 bits (pour diminuer la taille du code et augmenter la performance), elles doivent de plus être alignées. Le second étage (*Decode Instruction*) décode l'instruction et accède les registres impliqués, alors que le suivant (*EXEcute*) exécute toutes les opérations requises (calculs des adresses pour les chargements/rangements et toutes les opérations de l'UAL).

Les accès mémoire sont réalisés à l'étage 4 (*MEMory access*) et les registres mis à jour au dernier étage (*WriteBack*).

De plus, le Xtensa vient avec un jeu d'environ 80 instructions de bases et implémente un mécanisme de *Sliding Window*, c'est-à-dire qu'il possède un nombre de registres plus élevé que ceux visibles au programmeur. Au lieu de gaspiller des cycles d'horloge pour la sauvegarde et la restauration des registres à chaque appel et retour de fonction, la fenêtre des registres visibles est décalée lors d'un appel (fournissant ainsi un nouveau jeu de registres à la routine), puis replacée à sa position d'origine lors du retour. Ces deux fenêtres possèdent en plus une zone de recouvrement variable accélérant le passage de paramètres. Toutefois, malgré ces particularités, c'est son mécanisme de configuration qui différencie le Xtensa des processeurs standards.

Le processus de configuration débute en accédant la page Web de Tensilica [TENS03]. En utilisant un browser standard, le designer peut alors paramétrer le processeur souhaité [GONZ00]. Les possibilités varient de la technologie ciblée aux unités fonctionnelles additionnelles désirées (multiplicateur/accumulateur sur 16 bits, coprocesseur d'arithmétique flottante, etc.). Le Tableau 2.1 présente quelques-uns des paramètres à fixer. Pour chacun d'eux, des estimés de leur effet sur la puissance, la superficie et la fréquence d'utilisation du processeur sont disponibles et permettent au concepteur d'effectuer des choix éclairés tôt dans le processus de design. Le but est de concevoir le plus petit processeur possible, mais possédant tout le matériel nécessaire à l'exécution efficace de l'application ciblée.

Tableau 2.1 : Quelques paramètres configurables du Xtensa

Paramètres	Valeurs possibles
Taille des caches (instructions et données)	1, 2, 4, 8, 16 KO
Type des caches	À correspondance directe ou associative par 2, 3 ou 4; Écriture simultanée ou réécriture
Nombre de registres	16, 32
Dimension du bus externe	32, 64, 128 bits
Nombre d'interruptions	0-32
Niveaux des interruptions	0-6
Nombre de compteurs	0-3

Une fois le paramétrage du processeur terminé, le code RTL et les outils de développement logiciel sont automatiquement générés. Un modèle d'ISS est aussi fourni, permettant une simulation facile et rapide du processeur conçu. Dans le cas où les contraintes temporelles ne seraient pas respectées, Tensilica offre au concepteur l'opportunité d'ajouter des instructions additionnelles. Ces instructions, développées à l'aide du langage propriétaire TIE (*Tensilica Instruction Extensions*) similaire au Verilog, sont incorporées à l'étape de la configuration. Elles permettent de réaliser efficacement certaines tâches très spécifiques à une application, par exemple une manipulation de bits particulière ou encore de l'arithmétique spécialisée.

Ces nouvelles fonctionnalités génèrent alors du matériel. La Figure 2.3 présente ce matériel ajouté au pipeline du processeur, où la grande majorité de la logique additionnelle se retrouve au niveau du décodage des nouvelles instructions et à l'étage de l'exécution, où de nouvelles unités fonctionnelles sont alors nécessaires.

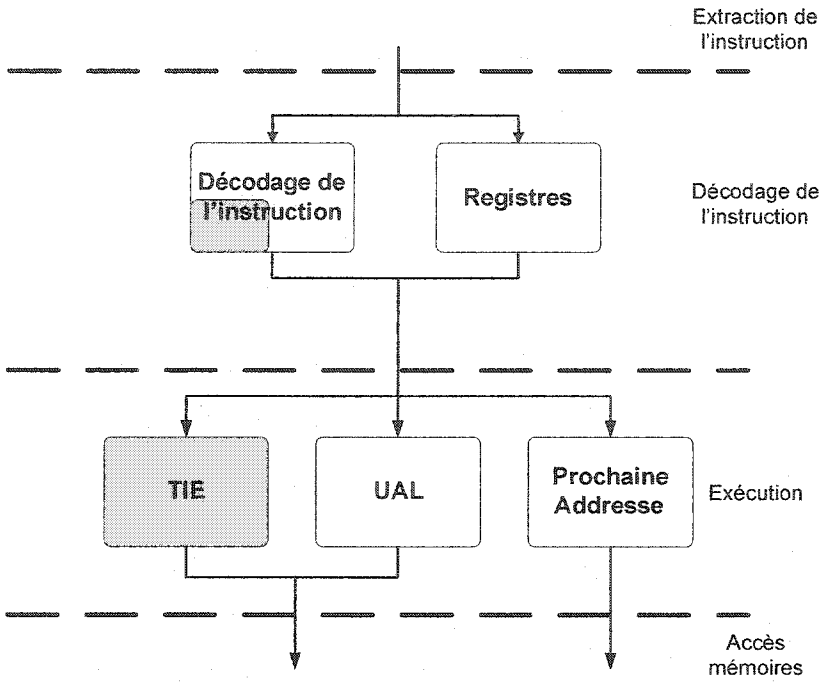

Figure 2.3 : Matériel généré par l'ajout d'instructions TIE (en foncé)

2.2.2. Intégration du Xtensa dans StepNP

D'abord, les mécanismes de configuration mis en place par Tensilica demeurent les mêmes peu importe la plate-forme de simulation retenue. De plus, comme le code RTL n'est pas considéré à l'étape de la recherche architecturale, seule l'intégration de l'ISS est alors nécessaire. Celle-ci comporte principalement deux tâches distinctes, soit l'intégration de l'ISS dans StepNP et l'ajustement du logiciel.

Intégration de l'ISS

Cette partie consiste essentiellement à insérer l'ISS du Xtensa dans un module SystemC [OURK02]. Il faut alors prendre bien soin de modifier adéquatement son interface de façon à respecter celle spécifiée dans StepNP. Cette partie a été réalisée en grande partie dans [LAVI04].

Modification du logiciel

Comme une suite d'outils de développement logiciel est fournie pour chaque nouveau processeur créé, aucune modification majeure du logiciel n'a à priori besoin d'être réalisée. Il suffit de recompiler l'application pour chaque version du processeur. Toutefois, certains ajustements doivent parfois être effectués, dépendamment des modifications apportées aux processeurs et à la plate-forme. Par exemple, une correspondance mémoire différente peut affecter le code. Certaines modifications des algorithmes doivent aussi être apportées pour l'utilisation efficace des instructions nouvellement ajoutées.

2.3. Modifications proposées à la méthodologie

L'approche actuelle du flot de co-design tend à répartir une application aux deux extrémités du spectre des solutions (Figure 2.2). Alors que l'ensemble de l'application est généralement réalisé en logiciel, les tâches plus critiques sont affectées à des coprocesseurs matériels. La méthodologie proposée s'inspire grandement de cette approche, exposant toutefois les opportunités supplémentaires engendrées par l'utilisation des processeurs configurables au niveau du partitionnement logiciel/matériel (Figure 2.1). Cette phase consiste maintenant en 4 étapes :

1. **Réalisation logicielle**

   L'application doit d'abord être réalisée entièrement en logiciel puisque le développement en est plus facile et plus rapide, particulièrement avec une plate-forme de développement comme Click. Le modèle conçu à l'étape de la spécification est généralement réutilisé, auquel une optimisation est effectuée. Le but n'est pas d'obtenir un code optimal en gagnant quelques cycles d'exécution, mais bien de s'assurer qu'aucune fonction n'est mal codée, ce qui aurait pour effet de fausser les résultats obtenus lors du profilage.

2. **Choix architecturaux**

   Après différentes analyses logicielles, la seconde étape consiste normalement à sélectionner le ou les processeurs les mieux adaptés à l'application ciblée. On choisit généralement un processeur d'usage général pour l'exécution des fonctions de contrôle, alors qu'on opte plutôt pour un processeur spécifique à un domaine d'applications pour l'exécution des tâches plus particulières.

   Dans le domaine des processeurs configurables, cette étape correspond à optimiser le ou les processeurs requis, c'est-à-dire choisir les meilleures caractéristiques architecturales possibles. La dimension des bus requis, le type et la taille optimale des caches ainsi que les unités fonctionnelles nécessaires sont tous des exemples de caractéristiques importantes à déterminer.

3. **Instructions spécialisées**

   Habituellement, si les résultats sont toujours insatisfaisants, on privilégie l'ajout de coprocesseurs matériels pour accélérer certaines tâches critiques et faire disparaître les goulots d'étranglement. Toutefois, dans bien des cas, il serait bien plus avantageux d'utiliser un processeur configurable et de lui ajouter des unités fonctionnelles. En effet, en plus de procurer des gains similaires aux coprocesseurs matériels, les unités fonctionnelles spécialisées apportent plusieurs avantages supplémentaires. D'abord, leurs temps de développement sont de beaucoup inférieurs à ceux des coprocesseurs et l'intégration avec le logiciel est triviale, puisqu'elle correspond simplement à l'ajout de nouvelles instructions. De plus, le surcoût des communications entre les processeurs et les coprocesseurs est évité, ce qui n'est pas négligeable compte tenu que l'un des principaux aspects limitatifs des SoC est la largeur de bande intra puce limitée. Finalement, cette approche est plus rapide et moins propice aux erreurs puisqu'une grande partie du processus est automatisée.

Il faut toutefois s'assurer que les instructions ajoutées n'auront pas d'impacts négatifs sur la fréquence d'utilisation du processeur. Pour ce faire, il est possible de synthétiser les instructions TIE réalisées et d'en mesurer le chemin critique.

4. **Ajout de coprocesseurs**

   Néanmoins, dans certains cas, l'usage de coprocesseurs se veut plus avantageux que l'ajout d'instructions spécialisées. Alors qu'une unité fonctionnelle n'accélère qu'une tâche, un coprocesseur en libère complètement le processeur, qu'il accélère la tâche ou non. De plus, un coprocesseur peut être partagé par plusieurs processeurs, facteur très intéressant pour les architectures multiprocesseurs. Différents motifs de design peuvent aussi influencer le choix de concevoir un coprocesseur, par exemple l'utilisation de différentes fréquences.

Toutes les autres étapes du flot de co-design (la spécification, le profilage, la simulation et la validation de l'architecture, la synthèse logicielle/matérielle, la génération du processeur et la cosimulation) demeurent les mêmes.

CHAPITRE 3
Exemples d'exploration architecturale

Afin d'évaluer les différents avantages des processeurs configurables dans la conception des processeurs réseaux, une plate-forme monoprocesseur composée d'un Xtensa a d'abord été modélisée et servira de structure de base pour la recherche architecturale. Par la suite, deux applications, spéculatives mais réalistes, ont été développées à l'aide de Click. La première spécification choisie est une application IPv4. Une telle application effectue généralement une grande variété de tâches provenant de l'ensemble des différentes catégories (voir section 1.2.4) et est donc très représentative du domaine des applications réseaux. La seconde, basée sur la première, introduit en plus certains algorithmes plus complexes provenant d'IPsec, dont le chiffrement DES.

Ces applications serviront de bancs d'essais et seront d'abord exécutées en entier par le Xtensa. Par la suite, la méthodologie de recherche architecturale proposée au Chapitre 2 sera appliquée. Les différents avantages des processeurs configurables, dont leur grande flexibilité et la possibilité d'ajouter des instructions, pourront alors être évalués pour chacune des applications. Les exemples présentés permettront aussi de clarifier et de démontrer la méthodologie proposée.

La recherche architecturale sera limitée à un seul processeur, pour lequel nous tenterons d'accélérer le traitement réalisé. Néanmoins, il serait possible de dupliquer le processeur pour obtenir la largeur de bande nécessaire en exploitant le parallélisme inhérent au traitement de paquets. Toutefois, une telle approche nécessite entre autres la mise à l'échelle du système d'interconnexions, un sujet non traité dans ce travail.

3.1. Plate-forme monoprocesseur basée sur un processeur configurable

La plate-forme initialement utilisée pour la recherche architecturale est similaire à celle présentée à la Figure 1.6a, pour laquelle les processeurs ARM sont remplacés par un modèle d'ISS adapté du Xtensa. Le système d'interconnexions employé est un canal de communication transactionnel. Au niveau du processeur, la latence des accès mémoire sur 32 bits est fixée à un cycle d'horloge. Il est possible d'ajuster différents paramètres de ce canal, comme la latence des accès et le nombre maximal de transactions par cycle d'horloge. Toutefois, comme ce travail ne porte pas sur l'optimisation ou l'étude des NoC, ces paramètres demeureront fixes. De plus, comme un seul processeur est utilisé, le coprocesseur de sémaphores a été retiré.

En plus du processeur, des mémoires et du canal de communication, des éléments supplémentaires sont nécessaires au transfert des paquets (ou des trames). Un mécanisme doit être défini pour permettre l'envoi de paquets à la plate-forme et la réception de ceux traités. Pour ce faire, un module SystemC nommé SPD (*Simple Packet Device*) est utilisé. La Figure 3.1 en présente la structure. Ce module possède deux interfaces similaires, soit une pour les paquets entrants, l'autre pour les paquets sortants.

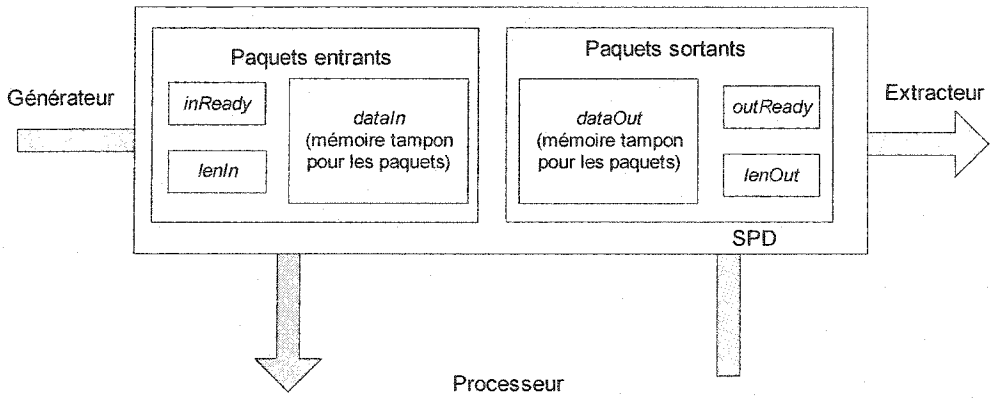

Figure 3.1 : Mécanisme d'échange des paquets

Les paquets envoyés à la plate-forme sont d'abord produits par un générateur de trafic (codé avec Click). Les étapes (processus de scrutation) devant être réalisées par celui-ci sont les suivantes :

1. le générateur vérifie continuellement la valeur du champ *inReady*. S'il vaut '1', alors un paquet non lu est déjà dans la mémoire *dataIn* et il doit réessayer;
2. lorsque *inReady* vaut '0', le générateur peut alors accéder le champ *dataIn* pour y placer un paquet;
3. il accède par la suite le champ *lenIn* pour y écrire la taille du paquet en octets;
4. puis écrit '1' au champ *inReady* pour préciser au processeur qu'un nouveau paquet a été enregistré dans la mémoire. Le processus reprend alors à l'étape 1.

De son côté, le processeur doit lui aussi suivre une certaine démarche (processus de scrutation) afin de récupérer les paquets laissés par le générateur :

1. lorsqu'il est prêt, le processeur vérifie la valeur du champ *inReady*. Si la valeur est de '1', alors un nouveau paquet est disponible. Sinon, il va lire continuellement cette valeur jusqu'à ce qu'un paquet soit prêt à être traité;
2. lorsque *inReady* vaut '1', le processeur lit alors la valeur du champ *lenIn* pour connaître la taille du paquet à lire;
3. il accède par la suite le champ *dataIn* de l'interface pour copier le paquet dans sa propre mémoire;
4. puis, lorsque la lecture du paquet est terminée, le processeur ajuste la valeur du champ *inReady* à '0' pour spécifier que le paquet courant à été récupéré et qu'un autre paquet peut maintenant être chargé.

Dès que le paquet est lu, le générateur en envoie un nouveau à la plate-forme. De son côté, aussitôt qu'il finit le traitement d'un paquet, le processeur en récupère un nouveau sur l'interface. Comme un paquet est toujours disponible, le débit de traitement maximal du processeur est toujours atteint. Pour la réception des paquets traités, un mécanisme similaire est utilisé.

3.2. Exemple d'exploration pour une application IPv4

Cette section présente les différentes étapes de la méthodologie de recherche architecturale proposée, appliquée à une application IPv4.

3.2.1. Description de l'application IPv4 (spécification)

La première étape consiste à réaliser une version logicielle de l'application avec Click. Comme il a déjà été spécifié, il suffit de choisir et d'interconnecter les bons éléments réseaux pour produire le comportement désiré. Les paquets circulent alors d'un élément à l'autre, jusqu'à ce que le traitement soit complet. La Figure 3.2 présente l'application IPv4 quelque peu modifiée d'une version originellement développée par le groupe de recherche de Click [CLIC03]. La Figure 3.3 présente l'environnement initial pour lequel elle avait été réalisée. Certains éléments ont dû être modifiés ou ajoutés pour tenir compte des particularités de la plate-forme, dont *SPDSource* et *SPDSink* pour appliquer les mécanismes de transfert des paquets définis à la section précédente.

Sommairement, cette application est conçue pour un routeur possédant deux interfaces, ETH0 et ETH1, mais pourrait aisément être généralisée à trois ou plus. Le routeur permet le transfert d'information entre deux réseaux locaux (LAN), dont l'un est rattaché à Internet, et permet la gestion du protocole ARP.

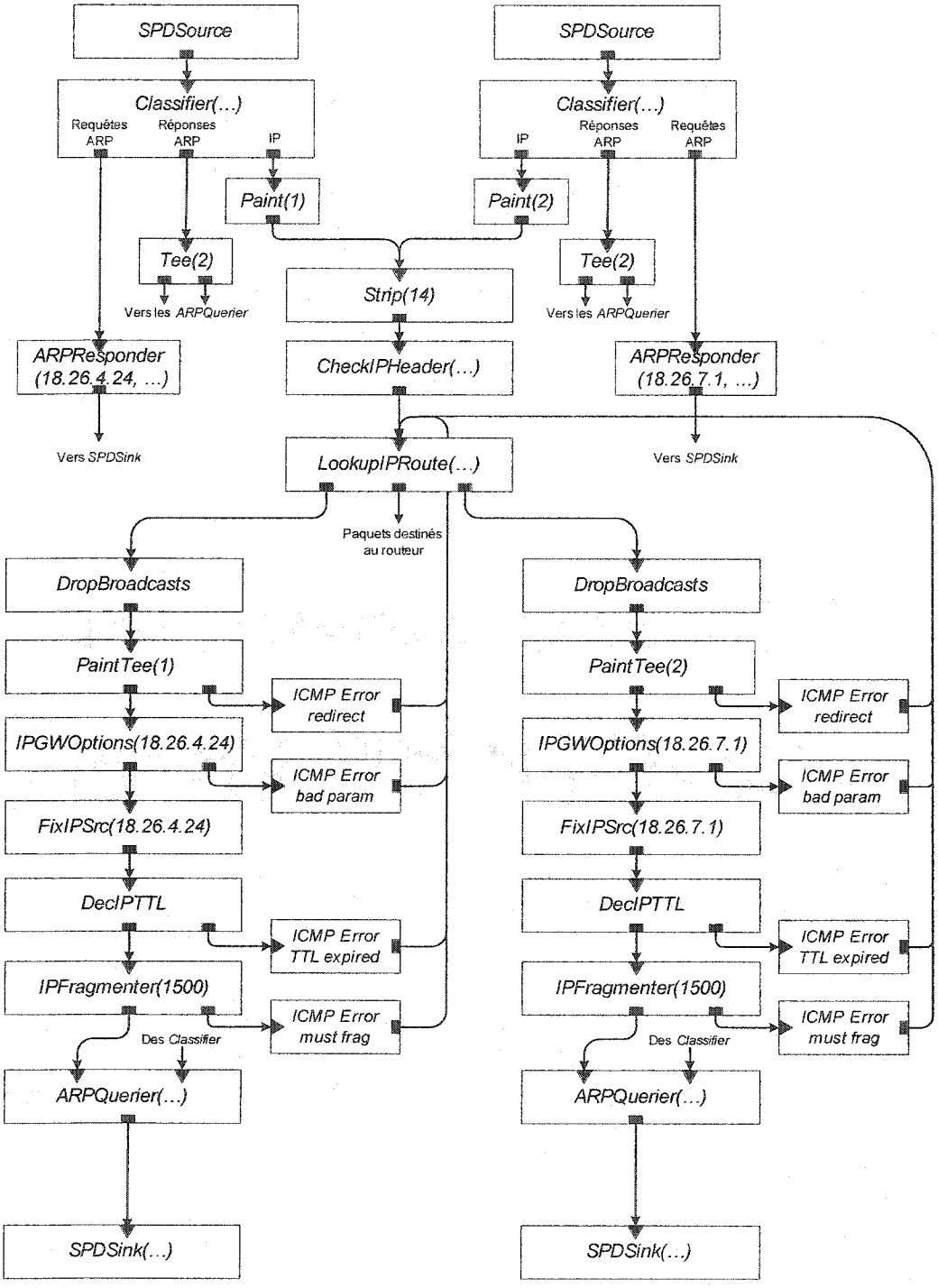

Figure 3.2 : Application IPv4

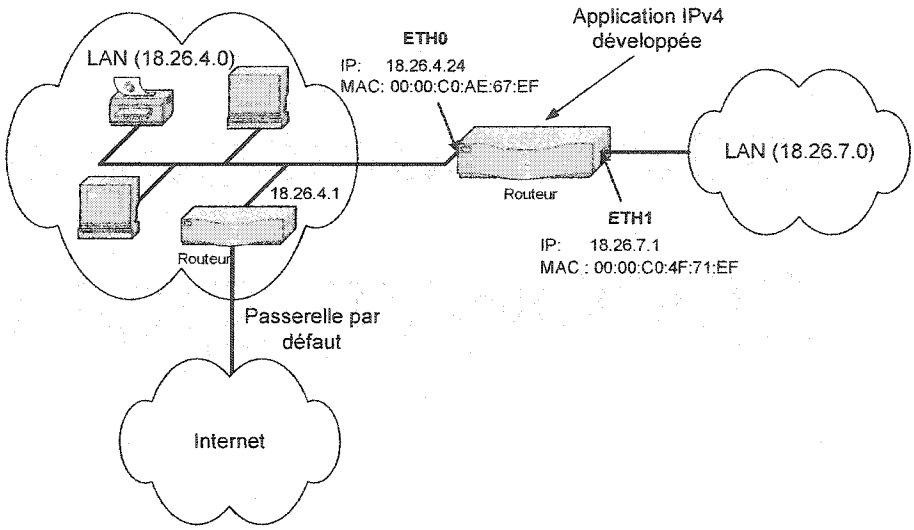

Figure 3.3 : Environnement de l'application IPv4 développée

Un premier élément, *SPDSource*, récupère d'abord les trames Ethernet envoyées à la plate-forme via les SPD. Ces trames sont par la suite séparées en trois catégories par un élément *Classifier*, soit celles effectuant une requête ARP, celles répondant à une requête ARP ou encore celles contenant un paquet IP. Toutes les autres sont rejetées. Classés selon leur type, les paquets parcourent alors un chemin distinct les uns des autres et subissent ainsi un traitement différent.

Les paquets ARP traversent un chemin plutôt simpliste. Les requêtes sont simplement traitées par l'élément *ARPResponder*, alors que les réponses sont envoyées aux éléments susceptibles d'avoir effectué la requête (*ARPQuerier*). Quant à eux, les paquets IP parcourent un chemin plus complexe. L'en-tête de chaque paquet doit d'abord être vérifié, puis séparé selon sa destination : soit l'une des deux interfaces ou encore le routeur lui-même. Dans ce dernier cas, les paquets adressés au routeur contiennent généralement des informations de contrôle, par exemple, des données pour la mise à jour des tables de routage. Dans le cas des processeurs réseaux, ces paquets sont couramment redirigés vers un processeur d'usage général à même le SoC ou encore connecté à celui-ci par une interface spécialisée. Dans notre cas, les paquets de contrôle de ce type sont

rejetés, tenant ainsi la table de routage fixe. De plus, comme l'application est relativement simple, la table de routage implantée ne contient que quelques entrées (voir l'ANNEXE A).

Par la suite, peut importe leur destination, les paquets reçoivent un traitement très similaire, incluant la décrémentation du champ TTL, la fragmentation, le traitement des options et la génération des messages ICMP en cas d'erreur.

Finalement, tous les paquets (ARP et IP) sont par la suite transmis à l'élément *SPDSink*, responsable de leur envoi à la bonne interface. Le fichier de configuration correspondant à l'application IPv4 est donné à l'ANNEXE B.

3.2.2. Profilage

L'application précédemment développée est d'abord exécutée en entier par un processeur Xtensa de base, dont les principales caractéristiques sont données au Tableau 3.1.

Tableau 3.1 : Principaux paramètres du Xtensa initial

Paramètres	Valeurs
Cache d'instructions	8KO, Associative par deux
Cache de données	8KO, Associative par deux
Largeur du bus de données	32 bits
Unités fonctionnelles spécialisées	Aucune
Nombre de registres disponibles	32
Compteur, interruption	Aucun

La plate-forme est alors alimentée grâce au générateur de trafic externe. Comme plusieurs protocoles n'opèrent que sur les en-têtes des paquets, la plus grande partie du traitement est souvent proportionnelle au nombre de paquets et non à leur taille. Pour ces types d'applications, l'industrie annonce couramment la performance de ses produits

pour des paquets de taille minimale [TSAI02]. Dans notre cas, une charge constituée que de trames Ethernet de 64 octets (sans préambule) est d'abord envoyée.

Toutefois, l'utilisation de paquets de taille minimale ou maximale ne fournit que les limites des débits atteignables. Pour disposer d'indications sur les performances réelles, un trafic plus réaliste, composé de paquets de différentes tailles incluant celle minimale et maximale, doit être utilisé. Pour Ethernet, [BRMC99] recommande d'utiliser des trames de 64, 128, 256, 512, 1024, 1280 et 1518 octets également distribuées. Cette charge constituera un deuxième trafic pour notre plate-forme. Dans les deux cas, tous les paquets transportés par les trames Ethernet sont des paquets IP. Aucun n'est erroné, ni ne nécessite de fragmentation.

Le profilage consiste alors à mesurer différentes caractéristiques du traitement des paquets, dont principalement les temps d'exécution requis. Le traitement total peut être distribué de plusieurs façons. Par exemple, on pourrait trouver pour chacune des tâches réseaux (voir section 1.2.4) le nombre de cycles dépensés. De tels résultats nous informeraient alors des tâches réseaux les plus coûteuses. Toutefois, comme le but est d'optimiser les goulots d'étranglement, il est préférable d'opter pour une distribution plus spécifique, reposant principalement sur l'exécution de fonctions de base ou encore le type d'instructions réalisées. Les catégories suivantes semblent bien convenir.

1. ***Window Overflow* et *Window Underflow* :** Ces routines sont appelées lorsque le mécanisme de *'Sliding Window'* ne peut s'exécuter correctement. Comme le processeur possède un nombre fini de registres, il peut arriver qu'il en manque. Dans ce cas, certains registres doivent alors être sauvegardés sur la pile, puis restaurés par la suite.
2. **Fonction *Push* :** Les éléments Click composant l'application s'échangent les paquets par l'appel de la fonction virtuelle *Push*. Cette fonction est donc appelée plusieurs fois par paquet.

3. **Gestion de la mémoire :** Cette catégorie inclut toutes les fonctions liées à la gestion de la mémoire (par exemple, *memcpy*, *malloc*, *memset*, *free*, etc.).
4. **Paquets IN/OUT :** Cette catégorie inclut les fonctions responsables du mécanisme d'échange des paquets entre le processeur et les interfaces.
5. **Vérification de l'en-tête IP :** Vérifie la validité des différents champs constituant l'en-tête IP.
6. **Checksum IP :** Calcul et mise à jour du champ Checksum de l'en-tête IP.
7. **Autres :** Inclut entre autres la consultation des tables, la fragmentation des paquets, le traitement des en-têtes Ethernet, etc.

Plusieurs outils peuvent être utilisés pour mesurer les différents temps d'exécution de l'application. Nous avons choisi d'employer le profileur fourni par Tensilica, puisque celui-ci est spécifiquement adapté au profilage du Xtensa.

Le Tableau 3.2 présente les temps d'exécution totaux (en cycles) et distribués obtenus lors du premier profilage. Comme les résultats le démontrent, il y a un prix non négligeable à payer sur la performance pour l'utilisation d'une application flexible de haut niveau. En effet, une telle application résulte généralement en une augmentation d'appels de fonctions imbriqués, et, dans notre cas, plusieurs appels de fonctions virtuelles. L'effet se répercute principalement sur les catégories *Window Overflow / Underflow* et *Fonction Push*, lesquelles prennent autant que 39.7 % du temps de traitement total pour des paquets de taille minimale, et 25% pour ceux de taille variable.

Tableau 3.2 : Résultats du profilage de l'application IPv4

Fonctions (Tâches)	Temps d'exécution			
	Taille minimale		Taille variable	
	Cycles	Pourcentage	Cycles	Pourcentage
Window Overflow et Underflow	823	22.8	892	14.6
Fonction Push	609	16.9	636	10.4
Gestion de la mémoire	898	24.9	3557	58.2
Paquets IN / OUT	173	4.8	190	3.1
Vérification de l'en-tête IP	123	3.4	73	1.2
Checksum IP	130	3.6	122	2
Autres	852	23.6	642	10.5
Exécution totale	3608	100.0	6112	100.0

Pour les paquets de taille minimale, le temps consacré au traitement des différentes catégories est grossièrement distribué. Cependant, pour les paquets de taille variable, la gestion de la mémoire prend une proportion très importante avec 58.2% du temps total d'exécution. En effet, alors que l'ensemble des tâches ne traitent que l'en-tête des paquets et ne dépendent aucunement de leur longueur, la gestion de la mémoire en est très affectée. Toute cette gestion s'explique d'abord par le fait que les paquets doivent en premier lieu être copiés dans la mémoire RAM du processeur avant tout traitement, puis redirigés aux interfaces à la fin. De plus, certains déplacements de paquets sont nécessaires en cours de traitement pour s'assurer que leur contenu demeure aligné avec la mémoire. Finalement, tous ces multiples déplacements deviennent plus laborieux à mesure que les paquets grossissent.

3.2.3. Exploration architecturale

Les résultats obtenus à la phase du profilage sont ensuite analysés et jumelés aux contraintes du projet pour guider efficacement les ingénieurs de systèmes dans l'étape cruciale de l'exploration architecturale. Cette étape consiste essentiellement à trouver des solutions efficaces pour optimiser l'application. Les moyens les plus économiques engendrant des temps de développement moindres seront retenus. Une méthodologie de recherche est d'ailleurs établie afin d'explorer en premier l'espace des solutions les plus susceptibles de satisfaire ces contraintes.

Cette section présente différentes alternatives architecturales obtenues en appliquant la méthodologie de recherche proposée. L'emphase est mise sur les possibilités engendrées par l'utilisation des processeurs configurables, lorsque possible. Les résultats intermédiaires sont résumés au Tableau 3.3 et au Tableau 3.4 de la section 3.2.3.5. Comme le temps d'exécution total de l'application IPv4 repose sur plus d'une tâche, plusieurs optimisations devront être réalisées pour obtenir des gains de performance globaux substantiels.

3.2.3.1. Optimisation logicielle

À cette étape, il est bien important de ne pas investir de ressources superflues dans la recherche de la perfection, mais simplement s'assurer d'être en possession d'applications convenablement codées. Bien que Click semble adéquat, il permet tout de même deux optimisations logicielles simples et rapides. La première consiste à ne compiler que les éléments utilisés par l'application, réduisant ainsi la taille de l'exécutable. Les effets sont une accélération de la phase d'initialisation du programme et possiblement une diminution du taux d'échecs des accès au cache d'instructions.

Pour s'échanger des paquets, tous les éléments Click dérivent d'une même classe '*Element*' et implémentent la fonction virtuelle *Push* qui y est définie. Il suffit alors aux éléments d'appeler cette fonction avec comme argument une référence au paquet transmis. Cette stratégie augmente grandement la flexibilité du système puisque n'importe quels éléments peuvent être interconnectés. Les appels de fonctions virtuelles sont toutefois résolus durant l'exécution (et non à la compilation) et nécessitent l'utilisation de tables de pointeurs de fonctions, engendrant une dégradation de la performance. La seconde optimisation est la dévirtualisation de la fonction *Push*. Elle consiste à effectuer une analyse syntaxique du code afin de remplacer les appels de fonctions virtuelles par des appels directs de fonctions. En effet, une fois la configuration établie, les éléments connaissent leurs voisins et l'ambiguïté peut être enlevée.

Ces deux méthodes d'optimisation logicielle ont été appliquées. Les gains obtenus sont donnés au Tableau 3.3 et au Tableau 3.4. Comme on pouvait s'y attendre, elles réduisent de quelques cycles le traitement effectué sur chaque paquet, accélérant principalement la catégorie *Fonction Push*.

3.2.3.2. Choix architecturaux

Un premier choix judicieux pour l'exécution d'applications réseaux est l'adoption d'un processeur gros-boutiste (*big endian*). En effet, comme les données transigées sur les réseaux sont gros-boutistes, les architectures petit-boutistes (*litle-endian*) requièrent une conversion des paquets ou une restructuration importante de leurs programmes.

Si l'on considère par exemple que seuls les en-têtes Ethernet (14 octets) et IP (20 octets minimums) doivent être convertis et qu'un appel de fonction est nécessaire par mot de 32 bits, alors 18 appels[1] sont tout de même requis pour la réception et l'envoi d'un seul paquet. Comme l'exécution de cette routine prend 30 cycles sur un processeur Xtensa, on économise donc en choisissant une architecture gros-boutiste environ 540 cycles par paquet. Il aurait toutefois été possible d'accélérer ces fonctions à l'aide d'instructions spécialisées ou encore de les réaliser par des coprocesseurs.

Une seconde remarque d'ordre général sur les applications réseaux est le nombre non négligeable d'accès mémoire de mots de 4, 8 et 16 bits. En effet, ce type d'applications nécessite souvent plusieurs comparaisons de champs de différentes tailles avec des expressions connues. Le processeur choisi devrait donc tenir compte de cette observation. Dans notre cas, le Xtensa possède déjà à la base des instructions spécialisées pour le chargement et le rangement de mots de 8 et 16 bits.

Plus spécifique à l'exécution de notre application, une première constatation est le nombre élevé des routines *window overflow* et *window underflow* réalisées. Une façon

---

[1] ((14 octets + 20 octets) / 4 octets) *2

facile et évidente d'y contrevenir est d'augmenter le nombre de registres disponibles. Le Xtensa permet de passer de 32 à 64 registres. Cet ajout a permis une diminution des appels de ces routines, accélérant cette catégorie d'environ 1.7 pour les deux types de paquets. Une seconde observation est le temps d'exécution requis pour la gestion de la mémoire. Toutefois, aucune option du Xtensa ne peut accélérer significativement cette catégorie.

L'ajout d'autres unités fonctionnelles n'a apporté aucun bénéfice visible. Aucune opération flottante, ni de multiplication / accumulation ne sont requises par l'application. Pour arriver à ces conclusions, une façon de procéder est d'abord de sélectionner les différentes unités fonctionnelles disponibles, puis vérifier grâce au profilage les gains encourus par ces modifications.

Finalement, l'optimisation de la taille des caches (d'instructions et de données) a été mise de côté dans ce travail. Bien qu'il soit important de dimensionner adéquatement les caches, ce processus itératif, basé sur la simulation, n'aurait apporté aucune information supplémentaire sur l'utilité des processeurs configurables. En effet, les même méthodes connues pour les processeurs standards auraient été appliquées.

3.2.3.3. Ajout d'instructions TIE

Tensilica propose, pour l'optimisation d'un programme, d'accélérer la fonction consommant le plus de temps par l'utilisation d'instructions TIE. Toutefois, comme le temps d'exécution de l'application IPv4 est partagé parmi certaines tâches, plusieurs fonctions doivent alors être identifiées, puis optimisées. Nous limitons ici l'exploration au calcul [BRBO88] et aux mises à jour [RIJS94] du champ Checksum des paquets IP pour illustrer le concept et les bénéfices rencontrés. La méthodologie pourrait toutefois être utilisée pour accélérer d'autres fonctions, par exemple l'ajout d'instructions de chargement et de rangement de mots de 4 bits pour la vérification du protocole.

Le Checksum doit d'abord être calculé pour les paquets entrants afin de vérifier l'intégrité de leur en-tête. L'algorithme consiste simplement à regrouper l'en-tête IP en mots de 16 bits, puis les sommer en complément 1. Sur une machine complément 2, la somme en complément 1 doit être calculée en additionnant les retenues, c'est-à-dire que tous les bits les plus significatifs qui débordent doivent être rajoutés aux bits les moins significatifs. L'implémentation utilisée par Click est donnée à la Figure 3.4 et consiste en deux étapes. D'abord, tous les mots de 16 bits formant l'en-tête sont normalement additionnés un à un en utilisant un accumulateur de 32 bits. Puis, les bits du débordement (les 16 bits les plus significatifs de l'accumulateur) sont sommés aux bits les moins significatifs.

```
uint32_t sum = 0;
uint16_t answer = 0;

/* Calcul de la somme en complément 2 */
while (nleft > 1)  {
    sum += *w++;
    nleft -= 2;
}

/* Addition des débordements */
sum = (sum & 0xffff) + (sum >> 16);
sum += (sum >> 16);

answer = ~sum;          /* tronquée à 16 bits */
return answer;
```

Figure 3.4 : Cœur de l'algorithme du Checksum

La somme ne peut efficacement être réalisée sur des mots de 32 bits, puisque le débordement doit être pris en compte. Pour surmonter cette limitation, nous avons ajouté une unité fonctionnelle capable d'effectuer l'addition en complément 1 d'un mot de 32 bits provenant de l'en-tête et d'un accumulateur sur 16 bits. Cette unité comporte

simplement deux additionneurs complément 1 en cascade, un premier pour l'addition des bits les moins significatifs avec les plus significatifs du mot de 32 bits, puis un second pour additionner le résultat avec l'accumulateur. La Figure 3.5 présente le nouvel algorithme utilisant l'instruction TIE développée. La première boucle s'exécute maintenant deux fois moins souvent puisque l'en-tête est lu par mots de 32 bits. De plus, le traitement supplémentaire nécessaire pour l'addition des débordements est retiré puisqu'il est maintenant réalisé au fur et à mesure.

```
while (nleft > 1)  {
   acc = IPCKSUM1(*w++, acc);
   nleft -= 4;
}

return ~acc;
```

Figure 3.5 : Algorithme du Checksum modifié par l'ajout d'une instruction TIE

Une fois l'intégrité de l'en-tête vérifiée, le champ du Checksum doit continuellement être mis à jour lorsque le paquet est modifié. Par exemple, l'algorithme précédent est utilisé lorsque l'adresse de destination est changée ou encore lorsque le paquet est fragmenté. Le Checksum est d'abord mis à 0 pour le calcul, puis remplacé par la valeur calculée. Une optimisation intéressante existe toutefois lors de la décrémentation du champ du TTL. Pour cette modification, la mise à jour du Checksum ne nécessite aucune lecture de l'en-tête, puisqu'on sait exactement quel nombre a été modifié, et de combien. Une instruction TIE a été ajoutée pour réaliser cette opération en un cycle d'horloge (voir [RIJS94] pour plus de détail sur l'algorithme).

Avec ces modifications, le Xtensa dispose maintenant d'instructions spécialisées pour le calcul et la mise jour du Checksum. Incluant les cycles nécessaires aux appels de fonctions, ces instructions permettent d'accélérer ces calculs d'environ 2.

3.2.3.4. Ajout de coprocesseurs

Avant tout traitement, le processeur Xtensa doit copier une trame Ethernet d'une interface à sa mémoire RAM. Une fois les données enregistrées dans son espace de travail (Figure 3.6a), le processeur peut ensuite traiter cette trame. Il s'agit d'abord de vérifier et de valider les différents champs du protocole du plus haut niveau. Par la suite, l'espace mémoire de l'en-tête Ethernet, généralement de 14 octets, est libéré et l'analyse du protocole suivant (Figure 3.6b) peut débuter.

Toutefois, comme l'en-tête est accédé par mots de 32 bits, des accès mémoire non-alignés surviendraient si aucune précaution n'était prise. En effet, pour la lecture du premier mot de l'en-tête IP et les lectures subséquentes, des accès requérants deux espaces mémoire accédés normalement par des adresses différentes seraient réalisés. Le Xtensa, tout comme les processeurs ARM et Sun, ne supportent pas ce type d'accès. Pour parer cette limitation matérielle, Click déplace simplement les paquets en mémoire de façon à toujours les garder alignés. Le résultat est présenté à la Figure 3.6c, où il est maintenant possible de lire le paquet IP par mot de 32 bits. Un procédé similaire est utilisé lorsque qu'un nouvel en-tête Ethernet est ajouté aux paquets sortants.

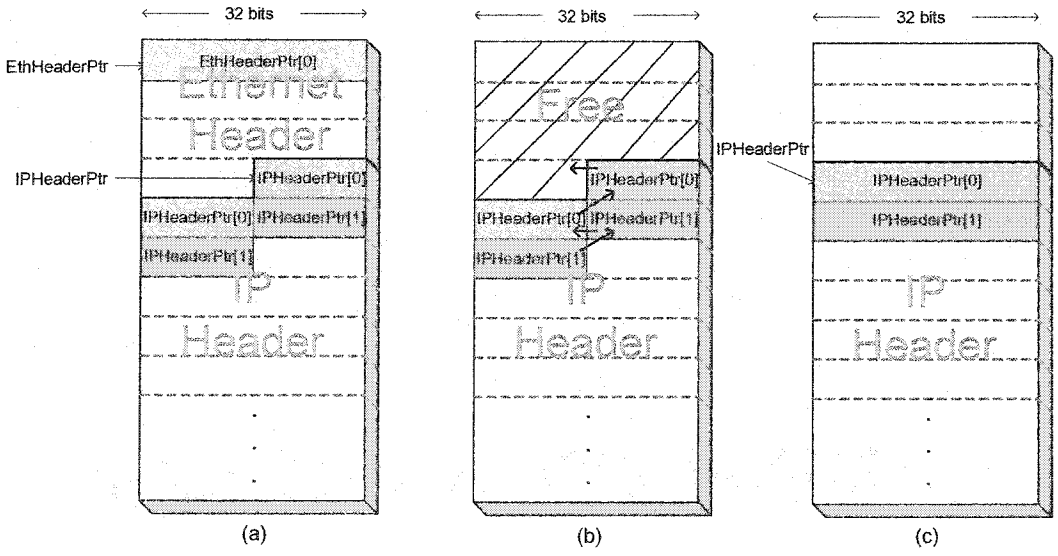

Figure 3.6 : Déplacement des paquets en mémoire pour l'alignement

Le déplacement des paquets pour l'alignement est toutefois réalisé de façon logicielle, octet par octet. Pour des paquets de taille minimale, cette tâche très coûteuse requiert actuellement 630 cycles et correspond à près de 18% du traitement total. Elle est d'ailleurs très perceptible au niveau du temps alloué à la gestion de la mémoire. Pour les paquets de taille variable, l'alignement prend 1,28 k cycles, correspondant cette fois à près de 21% du traitement.

La solution envisagée pour remédier efficacement à cette limitation est l'utilisation du *Xtensa Local Memory Interface* (XLMI) du Xtensa. Cette interface rapide est généralement utilisée pour connecter des périphériques, des coprocesseurs ou encore de la mémoire supplémentaire. Nous y avons d'abord connecté une mémoire de 4 ko et modifié l'application pour y enregistrer et traiter les paquets. Par la suite, nous avons ajouté la logique nécessaire pour supporter des accès non-alignés à cette interface. Les paquets peuvent donc maintenant être accédés de façon non-alignée et ainsi faire épargner au processeur un temps considérable.

Bien que cette simple modification soit très efficace, deux déplacements mémoire par paquet sont cependant toujours nécessaires, soit un déplacement du paquet dans la mémoire spécialisée connectée au XLMI, l'autre pour le diriger vers une interface de sortie à la fin du traitement. Même si cette tâche semble très simple, elle est tout de même très coûteuse pour le processeur. De plus, comme elle est principalement composée d'accès mémoire consécutifs, aucune instruction ne peut l'accélérer significativement. L'ajout d'un coprocesseur responsable du déplacement des paquets semble donc tout indiqué afin de décharger le processeur de ce fardeau.

Le mécanisme développé est très simple. Le processeur garde le contrôle de l'échange des paquets entre sa mémoire spécialisée et les interfaces. Toutefois, lorsqu'un déplacement de paquet est nécessaire, il transmet alors l'information à son coprocesseur

pour que celui-ci puisse réaliser le travail. Le coprocesseur reçoit alors l'adresse source des données à déplacer, l'adresse de destination ainsi que le nombre d'octets à transférer, puis effectue le déplacement désiré. Durant ce temps, le processeur libéré peut alors effectuer d'autres tâches plus complexes, dont le traitement d'un autre paquet. Noter qu'il aurait aussi été possible d'ajouter des modifications considérables au niveau du code pour ne déplacer de l'interface que l'en-tête du paquet, puis les données seulement si nécessaires.

Le support des accès non-alignés ainsi que l'ajout d'un coprocesseur permettent d'accélérer significativement la gestion de la mémoire, principalement pour les paquets de taille variable où un gain très intéressant d'environ 12.8 a été obtenu.

3.2.3.5. Résumé des gains engendrés par les différentes alternatives

Cette section résume les accélérations obtenues grâce aux différentes optimisations proposées. Celles-ci sont regroupées en quatre grandes classes, correspondant à chacune des étapes du partitionnement (section 2.3) : soit les optimisations logicielles, les choix architecturaux, l'ajout d'instructions TIE ou encore l'ajout de coprocesseurs. Le Tableau 3.3 donne les gains pour chacune des catégories du profilage lorsque le traitement est effectué sur des paquets de taille minimale, alors que les gains présentés au Tableau 3.4 ont été obtenus avec des paquets de taille variable. Toutes ces accélérations sont cumulatives, puisque le processus de l'exploration architecturale est itératif et qu'il n'y a aucune raison de ne pas utiliser les optimisations réalisées aux étapes précédentes.

Tableau 3.3 : Gains de chacune des optimisations pour des paquets de taille minimale

Fonctions (Tâches)	Gain pour chaque type d'optimisation			
	Logicielle	Config.	TIE	Coprocesseur
Window Over/Under flow	094	1.71	1.65	1.89
Fonction Push	1.21	1.18	1.19	1.31
Gestion de la mémoire	1.00	1.07	1.00	2.74
Paquets IN / OUT	1.03	1.08	1.15	1.08
Vérification de l'en-tête IP	1.34	1.42	1.37	1.40
Checksum IP	1.05	1.04	2.00	2.36
Autres	1.11	1.13	1.47	2.45

Tableau 3.4 : Gains de chacune des optimisations pour des paquets de taille variable

Fonctions (Tâches)	Gain pour chaque type d'optimisation			
	Logicielle	Config.	TIE	Coprocesseur
Window Over/Under flow	0.95	1.78	1.76	2.03
Fonction Push	1.11	1.10	1.04	1.40
Gestion de la mémoire	1.00	0.99	0.99	12.84
Paquets IN / OUT	1.13	1.13	1.18	1.17
Vérification de l'en-tête IP	1.09	0.82	1.00	0.99
Checksum IP	1.12	0.98	1.77	2.18
Autres	0.95	0.67	1.06	1.53

3.2.4. Simulation et validation

Entre chaque tentative architecturale, différentes simulations sont nécessaires afin de vérifier les fonctionnalités de la plate-forme. Néanmoins, les résultats de ces simulations sont réutilisés à l'étape du profilage et guideront nos choix quant aux prochaines alternatives possibles.

De plus, pour s'assurer de la validité des résultats, les nouvelles instructions TIE doivent être synthétisées au moins une fois. En effet, le délai de leur chemin critique doit être mesuré pour s'assurer qu'elles ne ralentissent pas le pipeline du processeur.

Finalement, une vérification doit aussi être apportée quant à la véracité des hypothèses faites sur les modules SystemC développés. Les performances spécifiées doivent être les plus réalistes que possible de façon à avoir des résultats valables. Pour les nouveaux modules, la synthèse est parfois la seule assurance. Toutefois, dans notre cas, comme le fonctionnement de notre module est très simple et ressemble fortement à celui d'un DMA (*Direct Memory Access*), ses performances sont vraisemblablement très réalistes (un accès mémoire par cycle d'horloge).

3.3. Exemple d'exploration pour une application IPsec simple

La sécurité des réseaux publics étant constamment mise à l'épreuve, le chiffrement apparaît comme une nécessité grandissante pour tous les appareils susceptibles de s'échanger de l'information. Basé sur des algorithmes complexes, le cryptage permet alors la transmission de données inintelligibles de tous, sauf du destinataire qui possède le moyen de restituer le message originel. Un algorithme largement utilisé en télécommunication est DES [NIST99] ou encore une amélioration nommée Triple-DES, consistant essentiellement à appliquer trois fois DES. Sa version améliorée est d'ailleurs un standard d'algorithmes de chiffrement utilisé par SSH (*Secure SHell*) et IPsec. Pour ces raisons, nous avons choisi d'incorporer à l'application IPv4 précédemment développée le chiffrement DES.

3.3.1. Description de l'application IPsec (spécification)

Basé sur IPv4, IPsec ajoute certains services pour sécuriser les communications aux travers les LAN, les WAN ou encore Internet. Afin de refléter plus justement les applications réseaux d'aujourd'hui et de demain, nous avons inclus à l'application précédente la possibilité de crypter les paquets IP via le service ESP en mode tunnel, normalement supporté par IPsec. La Figure 3.7 présente les ajouts apportés.

Les paquets IP entrants subissent initialement le même traitement qu'auparavant. Toutefois, avant d'être encapsulés dans une trame Ethernet et transmis à une interface, certaines opérations additionnelles sont effectuées sur ceux-ci. Un premier élément Click

vérifie d'abord si les paquets sont destinés à l'un des deux réseaux locaux (voir Figure 3.3). Dans un tel cas, les paquets demeurent intacts et sont simplement envoyés à l'élément *ARPQuerier*. Sinon, avant d'être redirigés vers l'extérieur, ceux-ci doivent alors être cryptés selon l'algorithme de DES. Pour se faire, un en-tête ESP est d'abord ajouté par l'élément *IPsecESPEncap*. Par la suite, les données sont cryptées par *IPsecDES*. Comme le mode tunnel est utilisé, un nouvel en-tête IP doit être ajouté aux paquets avant d'être finalement envoyés au *ARPQuerier*. Le fichier de configuration est donné à l'ANNEXE C.

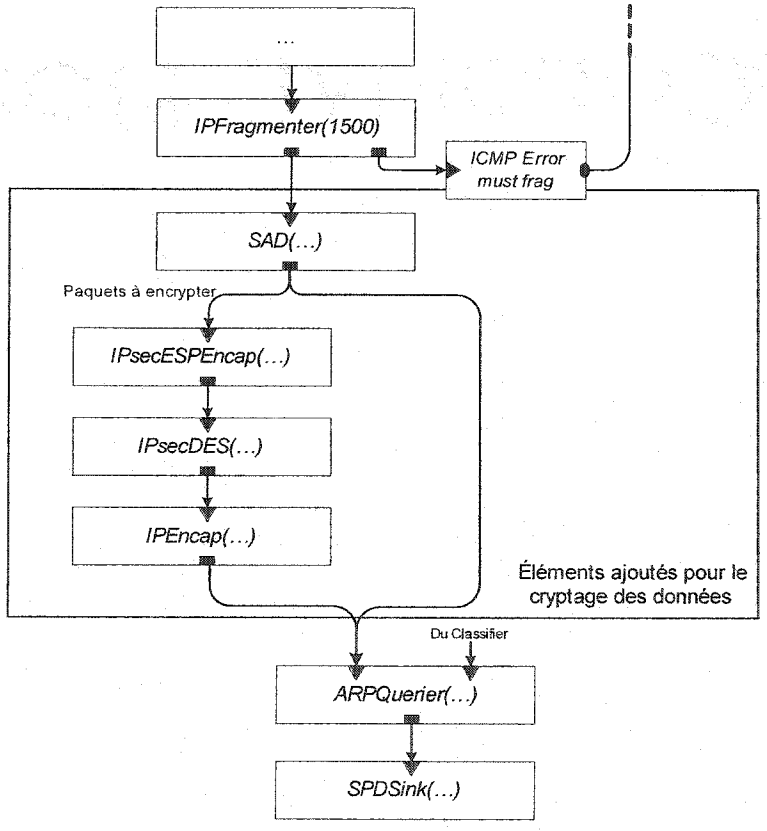

Figure 3.7 : Application IPv4 avec le chiffrement DES

Chiffrement DES

Le cœur de cette nouvelle application est son algorithme de chiffrement. Comme DES est très bien décrit dans la littérature, nous n'en présentons ici que les grandes lignes.

Cet algorithme est d'abord basé sur l'utilisation d'une même clef de 64 bits (en fait, 56 bits sont utilisés, les 8 autres servant à la parité) pour le cryptage et le décryptage des données. Elle doit donc être connue à la fois de l'émetteur et du récepteur. Différents mécanismes existent pour la génération et l'échange des clefs. Toutefois, dans ce travail, une clef unique sera utilisée pour le chiffrement de tous les paquets sortants du réseau local.

Les données sont traitées par blocs de 64 bits et l'algorithme consiste essentiellement en trois étapes, soit une permutation initiale, un traitement complexe dépendant de la clef utilisée, puis une permutation finale. Alors que les permutations initiales et finales se résument à la permutation des 64 bits du bloc à crypter selon un patron défini, le traitement intermédiaire est beaucoup plus complexe. D'abord, à l'aide de fonctions de permutation et de décalage, 16 clefs de 48 bits chacune sont générées. Par la suite, un traitement itératif est effectué sur les données. Une clef différente et 32 bits du bloc à crypter sont alors utilisés à chaque itération. Les opérations effectuées comportent entre autres l'adressage de tables, des fonctions logiques 'ou exclusives' et des permutations de bits.

Lorsque DES est utilisé seul, les blocs sont indépendants les uns des autres et chaque mot de 64 bits produit un mot unique de 64 bits. Une façon de rendre cet algorithme encore plus sécuritaire est de créer une interdépendance entre les différents blocs. Plusieurs solutions sont couramment utilisées. Celle retenue est le mode CBC (*Cypher Block Chaining*) qui consiste à insérer au début des données un nombre aléatoire codé sur 64 bits, puis réaliser entre les blocs voisins un 'ou exclusif' avant le chiffrement.

3.3.2. Profilage

Pour le profilage, la catégorie suivante a été ajoutée aux précédentes.

8. **Chiffrement DES-CBC :** Contient toutes les fonctions nécessaires au chiffrement DES-CBC.

Les temps d'exécution obtenus sont présentés au Tableau 3.5. Tous les paquets sont alors cryptés avant de quitter le routeur. Le travail supplémentaire effectué entraîne une augmentation du nombre de cycles nécessaires pour le traitement de différentes tâches. Alors que les catégories *Window Overflow / Underflow* et la gestion de la mémoire sont principalement affectées par les nouvelles fonctions de chiffrement, la fonction *Push* requiert plus de temps, compte tenu de l'augmentation du nombre d'éléments appelés par paquet.

Toutefois, même si le chiffrement n'est réalisé que par quelques éléments Click, il est de loin devenu la tâche la plus exigeante, réduisant significativement l'impact des autres sur la performance globale. De plus, comme le cryptage en mode tunnel est réalisé sur les paquets entiers, son temps d'exécution est grossièrement proportionnel à leur taille. Environ 48.9% du temps total est dépensé pour le cryptage des paquets de taille minimale alors qu'un trafic plus réaliste requiert plus de dix fois ce temps.

Tableau 3.5 : Résultats du profilage de l'application IPsec

Fonctions (Tâches)	Temps d'exécution			
	Taille minimale		Taille variable	
	Cycles	Pourcentage	Cycles	Pourcentage
Chiffrement DES-CBC	6177	48.9	68183	86.8
Window Overflow et Underflow	1443	11.7	1492	1.9
Fonction Push	1019	8.3	1131	1.4
Gestion de la mémoire	1769	14.3	5514	7
Paquets IN / OUT	201	1.6	204	0.3
Vérification de l'en-tête IP	94	0.8	94	0.1
Checksum IP	184	1.5	189	0.2
Autres	1457	12.9	1744	2.3
Exécution totale	12344	100.0	78551	100.0

3.3.3. Exploration architecturale

D'après les résultats du profilage obtenus, l'emphase de l'exploration architecturale doit être mise sur la recherche de solutions permettant d'accélérer le cryptage des paquets. Différentes améliorations sont proposées et leurs résultats intermédiaires sont résumés au Tableau 3.6 et au Tableau 3.7 de la section 3.3.3.5.

3.3.3.1. Optimisations logicielles

D'abord, comme elles sont simples et rapides, les optimisations logicielles précédemment réalisées pour l'application IPv4 ont été accomplies. Elles ont d'ailleurs permis l'économie de quelques cycles. De plus, comme la grande majorité du temps est utilisé pour le cryptage des données, nous avons examiné l'efficacité des algorithmes de chiffrement implantés. Comme l'implantation logicielle de DES utilisée par Click semble être l'une des plus rapides actuellement disponibles [YOUN92], aucune optimisation n'a toutefois pu être apportée à cet endroit.

3.3.3.2. Choix architecturaux

Bien sûr, comme l'application d'IPsec est basée sur une application d'IPv4, la discussion précédente sur les choix architecturaux demeure valide. Une architecture gros-boutiste de 64 registres semble donc tout indiqué pour débuter. Par la suite, il est important de vérifier si certains paramètres du processeur peuvent avoir un effet positif sur le chiffrement. Malheureusement, comme celui-ci n'est constitué que d'opérations mathématiques de bases, aucune unité fonctionnelle spécialisée n'accélère cette tâche.

3.3.3.3. Ajout d'instructions TIE

Le chiffrement semble un candidat idéal pour l'utilisation d'instructions TIE. En effet, bien que le cryptage DES prenne plusieurs milliers de cycles à s'exécuter sur le processeur, les fonctions effectuées demeurent assez simples et sont aisément réalisées en matériel. Par exemple, bien qu'il soit fastidieux d'effectuer la permutation de bits en logiciel, cette opération ne requiert en matériel que deux registres correctement reliés entre eux. Le décalage sur des nombres non conventionnels demande aussi plusieurs

cycles. Toutefois, en matériel, la dimension des registres est un paramètre aisément ajusté selon les besoins.

Quelques instructions TIE ont été réalisées pour remplacer l'exécution de certaines fonctions complexes [LAVI04]. Le programme a alors été modifié pour tenir compte de ce nouveau jeu d'instructions. Les gains obtenus pour le chiffrement sont très intéressants et varient entre 6,6 et 8, dépendamment de la taille des paquets utilisés. Une instruction spécialisée a aussi été ajoutée pour accélérer la génération des clefs. L'exécution de cette tâche est passée de 1900 cycles à 10 cycles, pour un gain très élevé de 190. Toutefois, comme la même clef est utilisée pour tous les paquets, cette fonction n'est appelée qu'une seule fois puisqu'il suffit alors de garder en mémoire les valeurs des clefs générées.

Les instructions TIE précédemment développées pour IPv4 (pour le calcul et la mise à jour du champ Checksum) ont aussi été ajoutées.

3.3.3.4. Ajout de coprocesseurs

Un coprocesseur aurait tout aussi bien pu être développé pour le cryptage des données. Toutefois, l'ajout d'instructions doit d'abord être privilégié. En effet, l'utilisation d'instructions spécialisées peut apporter des gains très intéressants sans pour autant prolonger significativement le temps de conception. De plus, l'ajout d'instructions n'entraîne aucune communication supplémentaire contrairement aux coprocesseurs. Finalement, les profilages subséquents démontrent que le chiffrement ne pourrait être accéléré de beaucoup puisqu'une grande partie des cycles restants ne sont pas proprement attribués aux fonctions de cryptage, mais plutôt à d'autres fonctionnalités dont l'ajout et la gestion des en-têtes ESP. Aucun coprocesseur pour le cryptage n'est donc nécessaire.

Avec le chiffrement maintenant accéléré, la gestion de la mémoire est redevenue une tâche importante du traitement. L'utilisation des coprocesseurs développés

précédemment pour l'alignement et le déplacement des paquets est donc de nouveau justifiée.

3.3.3.5. Résumé des gains engendrés par les différentes alternatives

Le Tableau 3.6 et le Tableau 3.7 résument les accélérations obtenues grâce aux différentes optimisations proposées pour des paquets de taille minimale et variable.

Tableau 3.6 : Gains de chacune des optimisations pour des paquets de taille minimale

Fonctions (Tâches)	Gain pour chaque type d'optimisation			
	Logicielle	Config.	TIE	Coprocesseur
Chiffrement DES-CBC	1.03	1.03	6.59	7.00
Window Over/Under flow	1.01	1.93	1.95	2.87
Fonction Push	0.95	0.98	1.24	1.42
Gestion de la mémoire	1.09	0.95	1.15	5.22
Paquets IN / OUT	1.01	1.01	1.03	1.06
Vérification de l'en-tête IP	1.00	0.93	0.99	1.01
Checksum IP	1.02	1.03	1.90	2.16
Autres	1.09	1.38	1.12	2.26

Tableau 3.7 : Gains de chacune des optimisations pour des paquets de taille variable

Fonctions (Tâches)	Gain pour chaque type d'optimisation			
	Logicielle	Config.	TIE	Coprocesseur
Chiffrement DES-CBC	1.01	1.00	7.96	7.93
Window Over/Under flow	0.97	1.91	1.94	2.93
Fonction Push	1.11	1.08	1.18	1.55
Gestion de la mémoire	1.09	1.04	1.10	9.83
Paquets IN / OUT	1.01	1.01	1.05	1.06
Vérification de l'en-tête IP	1.01	1.01	0.99	1.00
Checksum IP	0.97	1.02	1.85	2.22
Autres	0.78	1.06	1.13	2.99

3.3.4. Simulation et validation

Le fonctionnement des nouvelles instructions TIE a été vérifié en comparant les résultats obtenus avec des valeurs connues. [NIST98] propose un ensemble de valeurs d'entrée et

de clés pour lesquelles les résultats sont déjà calculés. De plus, la synthèse a permis de valider le chemin critique des différentes instructions.

CHAPITRE 4

Analyse des résultats

Alors que le chapitre 3 illustre la méthodologie proposée et l'effet des différentes optimisations sur chacune des catégories du profilage, ce chapitre présente d'abord les résultats globaux obtenus sur l'ensemble des applications. Par la suite, la validité de ces résultats est discutée compte tenu des limites du modèle actuel. Certaines améliorations sont alors proposées. Finalement, à la lumière des différents résultats obtenus, les avantages de l'utilisation des processeurs configurables sont abordés.

4.1. Synthèse des résultats

L'effet des différentes optimisations est ici évalué sous différentes perspectives, dont les débits et les latences atteints, les temps de simulation requis, etc.

4.1.1. Débit et latence

La Figure 4.1 et la Figure 4.2 présentent les gains globaux obtenus sur les temps d'exécution des deux applications. En appliquant la méthodologie proposée, IPv4 s'est vue accélérée de 1,92 son traitement de petits paquets et de 3,25 celui de paquets de taille variable. Ces accélérations s'élèvent à 3,57 et 6,92 pour l'application IPsec. Ces gains correspondent à la somme des optimisations partielles réalisées lors de l'exploration architecturale sur les différentes portions de l'application.

Alors que les optimisations logicielles et les choix architecturaux apportent des gains minimes, l'ajout d'instructions spécialisées et de coprocesseurs améliorent grandement la vitesse du traitement. Toutefois, comme l'application IPv4 repose sur plus d'une tâche, plusieurs instructions spécialisées supplémentaires devraient alors être réalisées pour accroître significativement les gains encourus.

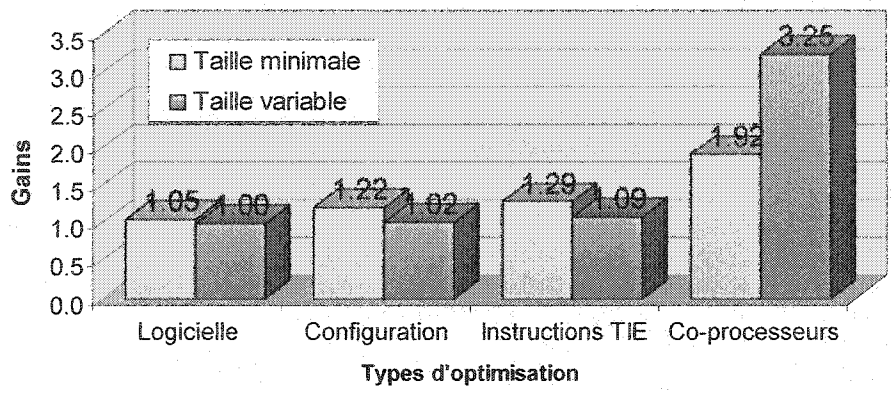

Figure 4.1 : Gains globaux pour l'application IPv4

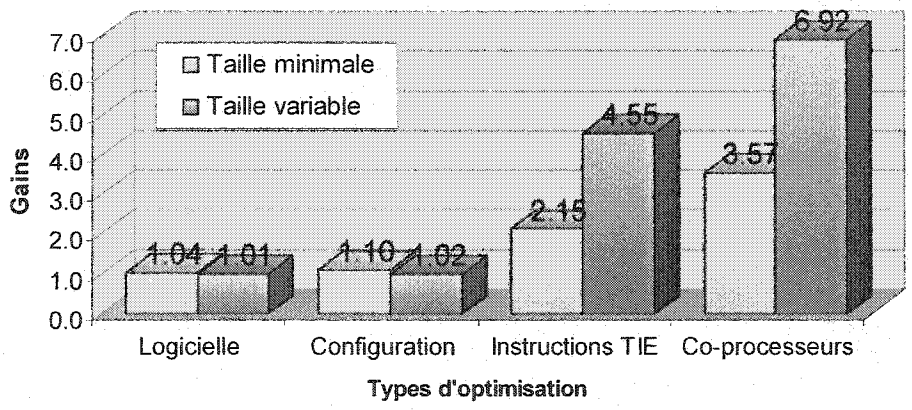

Figure 4.2 : Gains globaux pour l'application IPsec

Ces gains ont été mesurés à partir du nombre de cycles d'horloge requis par le processeur pour le traitement d'un paquet. Dans le cas du trafic plus réaliste, une moyenne a été calculée. Comme la fréquence du processeur est gardée constante malgré les ajouts apportés, aucun ajustement n'est alors nécessaire sur ces gains. Dépendamment des choix d'optimisation (puissance, fréquence ou superficie), un processeur Xtensa synthétisé avec une technologie typique de 0.13 µm et possédant les caractéristiques spécifiées peut être cadencé jusqu'à 400 MHz. Pour l'application IPv4, les débits maximums atteints sont donc de 109 Mbps pour le traitement de trames de 64 octets et de 1.16 Gbps pour des

trames de taille variable (dont la taille moyenne est d'environ 683 octets). Lorsque le cryptage DES est ajouté, ces débits diminuent à 57,1 Mbps et 193 Mbps.

La latence d'un routeur est le temps écoulé entre la réception d'un paquet et sa transmission. Puisque dans notre modèle une seule trame est traitée à la fois et qu'aucune contention ne survient à l'entré comme à la sortie, la latence correspond directement au nombre de cycles requis par le processeur pour son traitement. Des cycles supplémentaires doivent être ajoutés seulement lorsque le DMA est en fonction, puisque le processeur n'effectue alors qu'une partie du traitement. Ce coprocesseur prend deux cycles par mot de 32 bits pour la réception et l'envoi des trames. Pour des trames de taille minimale, 32 cycles doivent donc être ajoutés à la latence, alors qu'en moyenne 340 cycles additionnels sont requis pour les trames de taille variable. Malgré tout, le DMA permet tout de même de réduire considérablement la latence, puisque que le déplacement des trames nécessitaient auparavant plus de cycles de la part du processeur.

4.1.2. Temps d'initialisation et de simulation

Jusqu'à présent, les temps et les gains présentés font référence au traitement des paquets. Pourtant, avant d'être en mesure de débuter l'exécution de l'application, une phase d'initialisation est nécessaire. Click doit d'abord lire et traiter les paramètres passés au programme. Par la suite, il doit lire le fichier de configuration et effectuer une analyse syntaxique de celui-ci afin de déterminer les éléments utilisés ainsi que leur agencement. Finalement, chacun des éléments est crée et initialisé avec les paramètres spécifiés.

Bien qu'aucun effort n'ait explicitement été apporté pour accélérer la phase d'initialisation, les différentes optimisations la réduisent presque de moitié. La Figure 4.3 présente les temps d'initialisation (en nombre de cycles) de Click pour les deux fichiers de configuration utilisés. Différentes raisons expliquent ces résultats. D'abord, le nombre d'éléments disponibles est très déterminant puisqu'il dicte en partie le temps consacré à l'identification de ceux-ci. En effet, l'analyse syntaxique du fichier de configuration devient très laborieuse lorsque le nombre de possibilités augmente. C'est d'ailleurs

pourquoi la compilation minimale (optimisation logicielle) accélère de beaucoup la phase initiale. De plus, comme le fichier de configuration d'IPsec contient quelques éléments supplémentaires à IPv4, le temps requis pour son analyse est quelque peu plus élevé. Par la suite, l'augmentation du nombre de registres (configuration du processeur) accélère l'exécution générale du programme, y compris les différentes routines appelées lors du démarrage.

De leur côté, les instructions TIE réalisées ont un effet moins marqué puisqu'elles visent expressément le traitement des paquets. Seulement une instruction participe directement à accélérer l'initialisation. En effet, comme la clef de cryptage DES est fixe, l'instruction ajoutée pour la génération des clefs intermédiaires est réalisée à cette étape.

Finalement, l'ajout de coprocesseurs accélère aussi la phase d'initialisation. Comme expliqué précédemment, l'alignement est réalisé grâce à l'insertion d'éléments Click responsables du déplacement en mémoire des paquets. Lorsque les accès non-alignés sont supportés, ces éléments sont inutiles et le fichier de configuration est alors simplifié, accélérant ainsi l'initialisation.

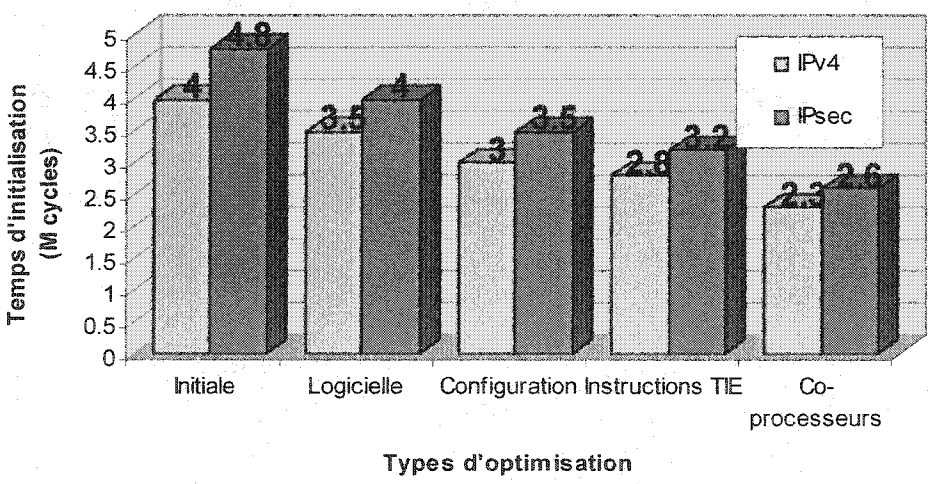

Figure 4.3 : Temps d'initialisation de Click

Comme plusieurs fonctions sont appelées autant à la phase de l'initialisation qu'à la phase du traitement, il est important de simuler suffisamment longtemps l'application afin de réduire l'effet de l'initialisation sur les résultats du profilage. Pour déterminer le temps de simulation nécessaire (en cycles d'horloge), la règle suivante a été fixée :

$$T_{traitement} > 100 * T_{initialisaiton}$$

C'est-à-dire que le traitement des paquets doit être d'au moins 100 fois supérieur au temps nécessaire à l'initialisation de l'application. Dépendamment du nombre de cycles requis pour l'initialisation et le traitement des paquets, des simulations variant entre 10 k et 100 k paquets ont été réalisées. Les temps réels des simulations étaient d'environ 6 heures sur un UltraSPARC-IIe avec 512 Mo de mémoire RAM.

4.1.3. Taille des exécutables

La taille des programmes pour les SoC est un aspect très important. Comme l'espace mémoire disponible sur une puce est limité, les programmes volumineux sont généralement placés à l'extérieur et seulement une portion du code se retrouve chargée en mémoire sur la puce. Comme les accès réalisés à l'extérieur sont généralement très coûteux, la taille des programmes peut diminuer dramatiquement ses performances, principalement pour les applications où l'exécution est très irrégulière (c'est-à-dire effectuant plusieurs sauts en mémoire).

Toutefois, comme notre plate-forme possède suffisamment d'espace mémoire sur la puce pour contenir tout le programme, cet effet n'est pas observé. La taille des exécutables n'affecte alors que les performances du cache d'instructions. La Figure 4.4 présente tout de même l'effet des différentes optimisations sur la taille des exécutables. Une réduction majeure est naturellement observée pour la compilation optimisée. Bien que l'ajout d'instructions TIE et de coprocesseurs permettent normalement une réduction du code en réalisant certaines fonctions complexes en matériel, leur effet est ici dissimulé. En effet,

une grande partie du programme correspond aux fonctions nécessaires à la phase d'initialisation.

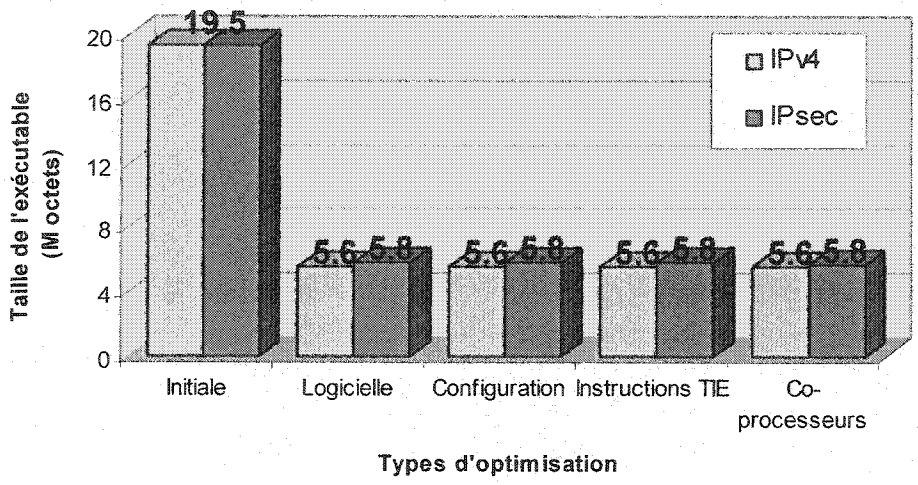

Figure 4.4 : Taille des exécutables

4.1.4. Autres

La performance est généralement la métrique prédominante utilisée lors de la recherche architecturale de systèmes à haute performance. Néanmoins, d'autres facteurs subjectifs (par exemple, la complexité et la flexibilité des designs) ou estimés (leur puissance dissipée et leur superficie requise) sont aussi pris en compte lors de l'évaluation des différentes alternatives.

Dans ce travail, nous avons limité l'étude sur les performances engendrées par les différentes solutions. Néanmoins, Tensilica fournit des estimés instantanés sur la fréquence, la puissance et la superficie des processeurs générés. Pour les autres constituants de la plate-forme, la puissance et la superficie peuvent être estimées par comparaison avec des modèles similaires déjà existants, en autre cas une synthèse sera nécessaire.

4.2. Limitations actuelles du modèle et améliorations possibles

D'abord, bien que les applications développées semblent dans l'ensemble représentatives du domaine des applications réseaux, elles demeurent relativement simples et correspondent à des versions abrégées d'applications réelles. Parmi les simplifications réalisées, on retrouve.

- **L'utilisation d'une petite table de routage :** Les applications développées utilisent une table de routage fixe ne contenant que quelques entrées. Comme plusieurs solutions existent déjà pour accélérer la consultation et le maintien de grandes tables de routage, dont l'usage de coprocesseurs [CLEA01b] ou même d'algorithmes utilisant des instructions spécialisées [TENS00], [TENS02], cette simplification avait été apportée pour accentuer la recherche d'autres sections de code critiques. L'intégration et l'évaluation des différentes solutions existantes pourraient toutefois s'avérer très intéressantes.

- **Aucune fragmentation :** Bien que la fragmentation soit actuellement possible avec les applications développées, la recherche a été limitée au cas sans fragmentation. Une prochaine étape pourrait donc regarder s'il est possible d'accélérer cette tâche, principalement avec l'usage d'instructions spécialisées.

- **Application de haut niveau :** Pour le développement de nos applications, Click a d'abord été retenu pour sa rapidité, puis pour sa flexibilité. Il demeure cependant une application de très haut niveau engendrant une perte de performance non négligeable, altérant ainsi quelque peu le profilage et dissimulant peut-être d'éventuelles sections critiques importantes. Toutefois, STMicroeletronics a développé une version allégée de Click mieux adaptée aux besoins des SoC. Une diminution très importante de la taille des exécutables et des gains de performance d'environ 25% ont été rapportés. Cette version, déjà intégrée à StepNP, devra éventuellement être portée sur le Xtensa.

- **IPsec :** Bien que la seconde application permet le cryptage DES de paquets IP via le service ESP en mode tunnel, elle ne réalise qu'une petite portion d'IPsec. Une attention particulière doit donc être apportée sur l'interprétation des résultats. En effet, l'objectif n'est pas de démontrer le débit soutenu par des processeurs configurables exécutant le protocole IPsec, mais plutôt d'illustrer comment ils peuvent accélérer des algorithmes complexes, dont le chiffrement.

En deuxième lieu, quelques simplifications ont aussi été apportées à la plate-forme. Premièrement, le modèle d'interconnexions utilisé est très simple et n'est pas très représentatif des latences normalement rencontrées pour les communications sur puce. Toutefois, comme notre plate-forme est constituée d'un seul processeur, l'impacte en est grandement réduit. Cependant, des modèles d'interconnexions plus fidèles à la réalité seront nécessaires pour la réalisation et l'évaluation de plates-formes multiprocesseurs, pour lesquelles la largeur de bande disponible sur puce est souvent un facteur déterminant.

Une deuxième simplification déjà discutée est la dimension de la mémoire du programme. En effet, une mémoire d'instructions pouvant contenir tout le programme a été utilisée. Une fois l'application chargée, aucune communication avec une mémoire externe n'est alors nécessaire, ce qui n'est pas toujours le cas pour des systèmes réels.

Finalement, certains points intéressants n'ont pas eu le temps d'être examinés ou ont simplement été laissés de côté :

- **Largeur du bus :** Une caractéristique intéressante du Xtensa est sa possibilité d'augmenter la largeur de son bus à 64 bits et même 128 bits. Par exemple, dans le cas du cryptage, il serait intéressant de regarder s'il est alors possible et avantageux d'accéder les paquets directement par blocs de 8 octets au lieu de 4.

- **Dimension de la taille des caches :** Aucune étude sur la taille des caches d'instructions et de données n'a été réalisée.

- **Architecture multiprocessus :** Comme l'utilisation de RISC multiprocessus à changement de contexte rapide semble être à la mode pour la réalisation des processeurs réseaux à haute performance (voir section 1.3.2.3), il serait intéressant d'évaluer comment doter le Xtensa de cet avantage.

4.3. Avantages des processeurs configurables

À plusieurs niveaux, les processeurs configurables et les processeurs embarqués d'usage général (ou encore les ASIP) semblent des alternatives sensiblement similaires pour la conception de SoC. D'abord, les processeurs configurables de base (sans aucune optimisation) obtiennent des performances comparables à celles obtenues avec des processeurs embarqués d'usage général [EEMB03]. De plus, dans tous les cas, les applications peuvent être développées dans un langage de haut niveau et utilisent une suite d'outils similaires.

Deuxièmement, comme une famille de processeurs est généralement disponible pour un modèle particulier, les processeurs embarqués offrent sensiblement les même possibilités au niveau de leur configuration que les processeurs configurables (taille et configuration des caches, architecture gros-boutiste ou petit-boutiste, etc.).

Finalement, comme pour des processeurs configurables, rien n'empêche une plate-forme basée sur des RISC embarqués standards d'utiliser des coprocesseurs pour accélérer certaines tâches. Ce qui distingue donc uniquement les processeurs configurables des autres, c'est leur possibilité d'ajouter des instructions spécialisées. Mais s'agit-il réellement d'un avantage ?

L'ajout de coprocesseurs permet d'obtenir des gains au minimum comparables à ceux réalisés par l'utilisation d'instructions spécialisées. Par exemple, le calcul et la mise à

jour du Checksum ainsi que le chiffrement DES auraient pu être accélérés avec l'usage de coprocesseurs. De plus, même si les coprocesseurs ne peuvent accélérer significativement une tâche, ils en déchargent au moins le processeur et peuvent même être partagés par plus d'un. Durant l'exploration architecturale, des modèles SystemC sont alors développés pour simuler les différents coprocesseurs désirés. Selon notre expérience, le temps de développement d'un modèle comportemental SystemC est très court et du même ordre de grandeur que la conception d'instructions spécialisées équivalentes. De plus, les deux nécessitent des modifications de l'application.

Toutefois, contrairement aux instructions spécialisées, les modèles SystemC requièrent souvent plusieurs étapes de raffinement pour valider les résultats et augmentent significativement le degré de complexité d'un design. Le temps de développement complet se voit donc prolonger de beaucoup, principalement aux étapes de la synthèse et des simulations post-synthèses. De plus, ils engendrent des communications supplémentaires avec le processeur, un facteur non négligeable compte tenu que l'un des principaux aspects limitatifs des SoC est la largeur de bande intra puce limitée.

Lors de la recherche architecturale, il est donc important de privilégier en premier l'utilisation d'instructions spécialisées avant l'usage de coprocesseurs. Cette remarque s'applique particulièrement bien au domaine des processeurs réseaux, puisque ces derniers contiennent généralement plusieurs coprocesseurs et que leur complexité ne cesse d'augmenter. Lorsqu'une tâche nécessite une accélération, l'effet de l'ajout d'unités fonctionnelles à même le processeur doit d'abord être évalué. Si les résultats sont satisfaisants, la recherche peut alors être déplacée vers d'autres tâches. Sinon, l'ajout d'un coprocesseur devra éventuellement être envisagé.

Conclusion et travaux futurs

Après avoir passé en revue le domaine des processeurs réseaux, dont leur environnement, leur architecture et les tâches qu'ils doivent accomplir, nous nous sommes dotés d'une méthodologie de partitionnement mieux adaptée à l'utilisation des processeurs configurables. En effet, comme l'objectif principal était d'évaluer, à haut niveau, les éventuels bénéfices de l'utilisation des processeurs configurables dans la conception des processeurs réseaux, certaines modifications à la méthodologie de partitionnement actuelle étaient nécessaires.

Globalement, le processus de recherche architecturale emprunté est le même que celui généralement utilisé. La première étape consiste à spécifier l'application à concevoir. Dans notre cas, une première version exclusivement logicielle est développée avec Click, puis compilée pour le Xtensa. La seconde étape consiste à profiler l'application. Les outils fournis avec le processeur configurable sont alors utilisés. L'étude des résultats du profilage permet ensuite d'identifier les fonctions à optimiser et de proposer certains partitionnements. Finalement, les performances de plusieurs alternatives sont normalement mises à l'épreuve et validées grâce à différentes simulations, clôturant ainsi la recherche architecturale. Ce processus est toutefois itératif et appliqué jusqu'à l'obtention de résultats valides et satisfaisants.

Traditionnellement, le partitionnement logiciel/matériel consiste à remplacer par du matériel les tâches d'une application devant être accélérées, pour lesquelles aucune optimisation logicielle n'est satisfaisante. Des coprocesseurs accomplissant les fonctionnalités désirées sont alors ajoutés. Ils peuvent être soit développés ou encore dérivés d'une bibliothèque de composantes existantes. Toutefois, l'arrivée des processeurs configurables dans la conception des SoC vient modifier les cartes. En effet, l'usage des processeurs configurables permet d'ajouter à l'étape du partitionnement, à

mi-chemin entre le logiciel et le matériel, une option intéressante. Cette alternative est l'utilisation d'instructions spécialisées.

L'ajout d'instructions spécialisées produit dans bien des cas des gains similaires aux coprocesseurs. En plus, comme les instructions spécialisées possèdent certains avantages sur ces derniers, elles doivent être privilégiées lors du partitionnement. D'abord, contrairement au coprocesseurs, elles sont aisément intégrées à l'architecture et au logiciel. Elles occasionnent donc des temps de développement minimes et sont moins propices aux erreurs. De plus, aucune communication supplémentaire n'est engendrée par leur utilisation et elles demeurent plus flexibles que les coprocesseurs.

Une fois cette méthodologie établie, nous nous sommes dotés d'une plate-forme permettant l'exploration rapide d'architectures. Notre choix s'est alors arrêté sur StepNP, puisque celle-ci était disponible et possédait déjà une bibliothèque de composants pour les processeurs réseaux ainsi qu'une suite d'outils pour le débogage et l'analyse de performance. Basée sur SystemC, StepNP facilite la conception et l'intégration de modules. Toutefois, comme elle ne comportait aucun processeur configurable, nous avons dû y intégrer le Xtensa de Tensilica. Cette tâche consistait essentiellement à insérer son ISS dans un module SystemC, tout en prenant soin de modifier adéquatement son interface de façon à respecter celle spécifiée dans StepNP. Une plate-forme monoprocesseur a finalement pu être développée sans aucune difficulté.

Par la suite, comme bancs d'essais, deux applications réseaux ont été développées avec Click. Actuellement le protocole le plus répandu pour les communications de niveau 3, une implémentation d'IPv4 a alors été choisie comme première spécification. Très utilisée, une telle application effectue en plus une grande variété de tâches réseaux, dont du filtrage, des consultations de tables, des calculs et des manipulations de données.

La deuxième application développée est basée sur la première, à laquelle nous avons ajouté le cryptage de paquets IP via le service ESP en mode tunnel, normalement supporté par IPsec. Cette application a alors permis de vérifier l'efficacité des processeurs configurables en présence d'algorithmes plus complexes reposants essentiellement sur la manipulation de bits.

Finalement, la méthodologie proposée a été appliquée pour chacune des deux spécifications. Alors que les optimisations logicielles et les choix quant à la configuration du processeur n'ont donné que des gains minimes, des accélérations très intéressantes ont été obtenues sans trop d'effort grâce à l'ajout d'instructions spécialisées et de coprocesseurs.

Nous avons pu démontrer que, dans certains cas réels, l'ajout d'instructions spécialisées peut accélérer significativement certaines tâches. Par exemple, pour le calcul et la mise à jour du Checksum ainsi que le cryptage DES, des gains entre 1,8 et 8 ont été obtenus pour ces tâches. De plus, ses ajouts ont été réalisés sans pour autant complexifier l'architecture du processeur réseau. Toutefois, dans certains cas, l'usage de coprocesseurs demeure indispensable. Le déplacement en mémoire de paquets est un excellent exemple. En effet, même si cette tâche est très simple, aucune instruction ne peut l'accélérer significativement car elle est principalement composée d'accès mémoire consécutifs. Dans ce cas, un coprocesseur s'avère très efficace en délivrant le processeur de cette tâche. Cependant, l'ajout de coprocesseurs occasionne en contre partie une complexité accrue du design.

Les travaux qui pourraient découler de ce projet sont nombreux. D'abord, l'intégration de la version allégée de Click devrait être effectuée en premier. L'utilisation de cette optimisation donnerait des résultats plus réalistes et surtout plus clairs en éliminant les cycles superflus engendrés par l'exécution d'une application de haut niveau. Par la suite, comme différentes simplifications ont été apportées dans ce travail, certaines tâches

réseaux n'ont pas été couvertes. On note notamment l'utilisation de tables de routage de grande taille et la fragmentation de paquets. D'un autre ordre d'idée, certains facteurs ont aussi été laissés de côté dont la largeur des bus et la dimension des caches.

Dans ce travail, le profilage de l'application et la détermination des instructions spécialisées ont été réalisés manuellement. Une avenue alternative et très prometteuse est l'automatisation de la génération des instructions [ATPL03]. Il serait donc intéressant de comparer nos résultats avec ceux produits par des outils automatisés. Bien que nous croyons l'optimisation manuelle d'algorithmes complexes plus efficace, il pourrait être intéressant de compléter l'optimisation du reste de l'application par la génération automatique d'instructions spécialisées.

Ces différentes améliorations apportées à la solution actuelle permettront l'obtention d'une plate-forme monoprocesseur beaucoup plus réaliste et complète. Toutefois, l'objectif principal de ce projet ne devra pas être perdu de vue. Comme tous les processeurs réseaux intègrent plusieurs processeurs, des architectures multiprocesseurs devront être développées.

Références

[AGER99a] AGERE INC. 1999. "The Challenge for Next Generation Network Processors". 7p. http://home.earthlink.net/~ilazar/agere.pdf (Page consultée le 28 novembre 2003)

[AGER99b] AGERE INC. 1999. "Building Next Generation Network Processors". 8p.

[ARCO01] ARNOLD, M., CORPORAAL, H. 2001. "Designing Domain-Specific Processors". *Proceedings of the Ninth International Symposium on CODES*. Copenhagen, Denmark. p. 61-66.

[ATPL03] ATASU, K., POZZI, L., LEENE, P. 2003. "Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints" *DAC 2003*. California, USA. p. 256-261.

[BRBO88] BRADEN, R., BORMAN, D. 1988. "Computing the Internet Checksum" (RFC 1071). 24p. http://www.ietf.org/rfc.html

[BREC03] Voir le site Web de Brecis Communication : http://www.brecis.com/

[BRMC99] BRADNER, S., MCQUAID, J. 1999. "Benchmarking Methodology for Network Interconnect Devices" (RFC2544). 31p. http://www.ietf.org/rfc.html

[CADE03] Voir le site Web de Candence : http://www.cadence.com/

[CBRB04] CHEVALIER, J., BENNY, O., RONDONNEAU, M., BOIS, G., ABOULHAMID, E. M., and BOYER, F.-R. 2004. "SPACE: A Hardware/Software SystemC Modeling Platform Including an RTOS". *Design & Verification Conference (DVCon)*. San Jose, USA.

[CHAO02] CHAO, H. J. 2002. "Next Generation Routers". *Proceedings of the IEEE,* 90:9. 1518-1558.

[CHMO01] CHEN, B., MORRIS, R. 2001. "Flexible Control of Parallelism in a Multiprocessor PC Router". *Proceedings of the USENIX 2001 Annual Technical Conference*.Boston, USA. 14p.

[CISC03] Cisco Systems INC. "Cisco 2500 Series Routers". http://www.cisco.com/en/US/products/hw/routers/ps233/index.html (Page consultée le 25 novembre 2003)

[CLEA01a] ClearSpeed™. 2001. "A 40 Gbit/s Network Processor Design Platform" A White Paper for Network System Designers. 12p. http://www.clearspeed.com/

[CLEA01b] ClearSpeed™. 2001 "The ClearSpeed™ Table Lookup Engine" A White Paper for Network System Designers. 8p. http://www.clearspeed.com/

[CLIC03] Voir le site Web de Click : http://www.pdos.lcs.mit.edu/click/

[CLNP02] CESARIO, W. O., LYONNARD, D., NICOLESCU, G., PAVIOT, Y., YOO, S., JERRAYA, A. A., GAUTHIER, L., and DIAZ-NAVA, M. 2002. "Multiprocessor SoC platforms: a component-based design approach". *Design & Test of Computers, IEEE*. 19:6. 52-63.

[COWA03] Voir le site Web de Coware : http://www.coware.com/

[DESL04] DESLAURIERS, François. 2004. Mémoire. *Création d'un environnement de test pour l'évaluation de performance de réseaux intégrés sur puce*. École Polytechnique de Montréal. (mémoire en préparation)

[DUCH03] DUTT, N. and CHOI, K. 2003. "Configurable Processors for Embedded Computing". *Computer*. 36:1. 120-123.

[EELS97] EGGERS, S. J., EMER, J. S., LEBY, H. M., STAMM, R. L., and TULLSEN, D. M. 1997. "Simultaneous multithreading: a platform for next-generation processors". *Micro, IEEE*. 17:5. 12-19.

[EEMB03] Voir le site Web du Embedded Microprocessor Benchmark Consortium : http://www.eembc.org/

[EZCH03] Voir le site Web de EZchip : http://www.ezchip.com/

[GNU03] Voir le site Web de GNU : http://www.fsf.org.

[GONZ00] GONZALEZ, R. E. 2000. "Xtensa: A Configurable And Extensible Processor". *IEEE Micro*. 20:2. 60-70.

[HEPA90] HENNESSY, J. L., PATTERSON, D. A. 1990. "Computer Architecture a Quantitative Approach". 2th ed. San Francisco. *Morgan Kaufmann Publishers*. 760p.

[HLWW02] HAVERINEN, A., LECLERCQ, M., WEYRICH, N., and WINGARD, D. 2002. "SystemC™ based SoC Communication Modeling for the OCP™ Protocol". 39p. http://www.ocpip.org/data/ocpip_wp_SystemC_Communication_Modeling_2002.pdf (Page consultée le 28 novembre 2003)

[IBM00] IBM Microelectronics Division. 2000. "IBM Processor for Network Ressources : A Cell/Frame Network Processor for Low Bandwidth Network Processing Applications -- Under 1 Gbps". http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF77852569880077B441/$file/npr.wp.pdf (Page consultée le 28 novembre 2003)

[IBM03] Voir le site Web de IBM : http://www.ibm.com/

[INTE03] Voir le site Web de INTEL : http://www.intel.com/

[KEAT98a] KENT, S., ATKINSON, R. 1998. "Security Architecture for the Internet Protocol (RFC 2401)". 66p. http://www.ietf.org/rfc.html

[KEAT98b] KENT, S., ATKINSON, R. 1998. "IP Authentication Header (RFC 2402)". 22p. http://www.ietf.org/rfc.html

[KEAT98c] KENT, S., ATKINSON, R. 1998. "IP Encapsulating Security Payload (RFC 2406)". 22p. http://www.ietf.org/rfc.html

[KOHL01] KOHLER, E. 2001. *The Click Modular Router*. Thèse de doctorat. Massachusetts Institute of Technology. 127p. http://www.pdos.lcs.mit.edu/papers/click:kohler-phd/thesis.pdf (Page consultée le 28 novembre 2003)

[LAVI04] LAVIGUEUR, Bruno. 2004. Mémoire. *Utilisation de jeux d'instructions spécialisés afin d'optimiser une application réseau*. École Polytechnique de Montréal. (mémoire en préparation)

[MABA03] MANNION, Paul, BARTLEY, Ian. 2003. "Fastpath Processing in the Internet Core". 7p. http://www.s3group.com/pdf/CDC2001-FastpathProcessing.pdf (Page consultée le 28 novembre 2003)

[MIHA02] MIHAL, A., KULKARNI, C., MOSKEWICZ, M., TSAI, M., SHAH, N., WEBER, S., Jin, Y., KEUTZER, K., SAUNER, C., VISSERS, K., and MALIK, S. 2002. "Developing Architectural Platforms: A Disciplined Approach". *Design & Test of Computers, IEEE* . 19:6. 6-16.

[MOTO03] Voir le site Web de Motorola : http://www.motorola.com/

[NETL00] NetLogic Microsystems Inc. 2000. "High Performance Layer 3 Forwarding : The Need for Dedicated Hardware Solutions". 6p. http://www.netlogicmicro.com/pdf/CIDR_white_paper.pdf (Page consultée le 28 novembre 2003)

[NIST98] NIST SP 800-17. 1998. "Modes of Operation Validation System (MOVS): Requirements and Procedures" 153p. http://csrc.ncsl.nist.gov/publications/nistpubs/800-17/800-17.pdf. (Page consultée le 28 novembre 2003)

[NIST99] NIST FIPS Pub. 1999. "Data Encryption Standard (DES)". http://www.itl.nist.gov/fipspubs/fip46-2.htm (Page consultée le 28 novembre 2003)

[OURK02] OUSSOROV, L., RAAB, W. H. U., and KRAVTSOV, A. 2002. "Integration of Instruction Set Simulators into SystemC High Level Models". *Proceedings of the Euromicro Symposium on Digital System Design*. p. 126-129.

[PAPB02] PAULIN, P. G., PILKINGTON, C., BENSOUDANE, E. 2002. "StepNP: a system-level exploration platform for network processors". *Design & Test of Computers, IEEE*. 19:6. 17-26.

[PMCS99] PMC-Sierra Inc. 1999. "A New Architecture for Switch and Router Design". 8p. http://www.pmc-sierra.com/pressRoom/pdf/lcs_wp.pdf (Page consultée le 28 novembre 2003)

[QLBA04] QUINN, D., LAVIGUEUR, B., BOIS, G., ABOULHAMID, M. 2004. "A System Level Exploration Platform and Methodology for Network Applications Based on Configurable Processors". *Design Automation and Test in Europe, DATE'04*. Paris, France. 6p.

[RIJS94] RIJSINGHANI, A. 1994. "Computing of the Internet Checksum via Incremental Update" (RFC 1624). 6p. http://www.ietf.org/rfc.html

[SHAH01] SHAH, Niraj. 2001. *Understanding Network Processors*. Master's thesis, University of California, Berkeley. 89p.

[SPKP00] SPALINK, T., KARLIN, S., and PETERSON, L. 2000. "Evaluation Network Processors in IP Forwarding". TR-626-00. Princeton University. 12p. http://www.cs.princeton.edu/nsg/router/papers/ixp.pdf (Page consultée le 28 novembre)

[SPUR03] SPURGEON, Charles. "Charles Spurgeon's Ethernet Web Site". http://www.ethermanage.com/ethernet/ethernet.html

[STAL99] STALLINGS, William. 1999. "Data & Computer Communications". 6th ed. *Prentice Hall*. 810p.

[TENS00] Tensilica Inc. 2000. "Xtensa Processor Extensions for Fast IP Packet Forwarding". 30p. http://www.tensilica.com/

[TENS02] Tensilica Inc. 2002. "Xtensa Processor Extensions for Fast IP Packet Classification". 32p. http://www.tensilica.com/

[TENS03] Voir le site Web de Tensilica : http://www.tensilica.com/

[TSAI02] TSAI, M. 2002. "Network Processor Benchmarks : Methodologies and Techniques for Network Processor Evaluation". 82p.

[VITK00] VIBHATAVANIJ, K., TZENG, N.-F., KONGMUNVATTANA, A. 2000. "Simultaneous multithreading-based routers". *Proceedings of 2000 International Conference on Parallel Processing*. Toronto, Canada. p. 362-369.

[VSVN02] VILLARREAL, J., SURESH, D., VAHID, F., NAJJAR, W. 2002. "Improving Software Performance with Configurable Logic". *Design Automation for Embedded Systems*. 7:4. 325-339.

[YIPE02] YITZCHAK, G. and PETERSON, L. 2002. "A Comparative Study of Extensible Routers". *Open Architectures and Network Programming Proceedings*. p. 51-62.

[YOUN92] YOUNG, Éric. 1992. "DES source code (DES-YOUN.ZIP)". http://www.owlriver.com/test/disk3/readme.txt

Annexes

ANNEXE A
Contenu de la table de routage

Le Tableau A-1 détaille les entrées de la table de routage implantée. Comme l'algorithme du '*longest prefix match*' est employé, un masque est utilisé pour identifier le nombre de bits valides des adresses. De plus, pour chaque entrée, un champ spécifie la sortie correspondante de l'élément 'LookupIPRoute', vers laquelle le paquet doit être dirigé. Les éléments subséquents déterminent alors quel traitement doit être réalisé sur les paquets en fonction de leur destination. Sommairement, la table de routage est la suivante.

- **Sortie 0:** Tous les paquets destinés au router.
- **Sortie 1:** Tous les paquets destinés au LAN 18.26.4 ou tous les autres destinés au WAN. Pour ces derniers, l'adresse de la passerelle est 18.26.4.1.
- **Sortie 2 :** Tous les paquets destinés au LAN 18.26.7.

Adresse	Masque	Prochain saut	Sortie de l'élément
18.26.4.24	32	-	0
18.26.4.255	32	-	0
18.26.4.0	32	-	0
18.26.7.1	32	-	0
18.26.7.255	32	-	0
18.26.7.0	32	-	0
18.26.4.0	24	-	1
18.26.7.0	24	-	2
0.0.0.0	24	18.26.4.1	1

Tableau A-1 : Entrées de la table de routage

ANNEXE B
Fichier de configuration pour IPv4

```
// ipv4.click

// This file is a network-independent version of the IP router
// configuration taken on www.pdocs.lcs.mit.edu/click/

// We use an SpdSource and an SpdSink click elements to transfer
// packets from the platform to the interface (SPD)

// eth0, 00:00:C0:AE:67:EF, 18.26.4.24
// eth1, 00:00:C0:4F:71:EF, 18.26.7.1

// Outputs of the Classifiers:
// 0. ARP queries
// 1. ARP replies
// 2. IP
// 3. Other
// We need separate classifiers for each interface because
// we only want proxy ARP on eth0.
c0 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -);

idle::Idle -> [0]c0;                    // Just use the ETH1 for the moment

spdsource::SpdSource(ADDR 0x6fff0000)   //
// -> Print(Packet_received_on_Xtensa, 0)
    -> [0]c1;

out0 :: SpdSink(ADDR 0x6fff0000); // Should use two SPD for each ETH (0 and 1)
out1 :: SpdSink(ADDR 0x6fff0000);
to1 :: Discard;                 // to1 means to Linux or to the control plan.

// An "ARP querier" for each interface.
```

```
fake_arpq0 :: EtherEncap(0x0800, 00:00:c0:ae:67:ef, 00:00:c0:4f:71:ef); //ARPQuerier
(18.26.4.24, 00:00:C0:AE:67:EF);
fake_arpq1 :: EtherEncap(0x0800, 00:00:c0:4f:71:ef, 00:00:c0:4f:71:ef); //ARPQuerier
(18.26.7.1, 00:00:C0:4F:71:EF);

// Deliver ARP responses to ARP queriers as well as Linux.
t :: Tee(3);                    // Tee(2) should be enough !!!
c0[1] -> t;
c1[1] -> t;
t[0] -> Discard;
t[1] -> fake_arpq0; // was -> [1]arpq0
t[2] -> fake_arpq1; // was -> [1]arpq1

// Connect ARP outputs to the interface queues.
fake_arpq0 -> out0;
fake_arpq1 -> out1;

// Proxy ARP on eth0 for 18.26.7, as well as cone's IP address.
ar0 :: ARPResponder(18.26.4.24 00:00:C0:AE:67:EF,
                    18.26.7.0/24 00:00:C0:AE:67:EF);
c0[0] -> ar0 -> out0;

// Ordinary ARP on eth1.
ar1 :: ARPResponder(18.26.7.1 00:00:C0:4F:71:EF);
c1[0] -> ar1 -> out1;

// IP routing table. Outputs:
// 0: packets for this machine.
// 1: packets for 18.26.4.
// 2: packets for 18.26.7.
// All other packets are sent to output 1, with 18.26.4.1 as the gateway.
rt :: StaticIPLookup(18.26.4.24/32 0,
                     18.26.4.255/32 0,
                     18.26.4.0/32 0,
                     18.26.7.1/32 0,
                     18.26.7.255/32 0,
                     18.26.7.0/32 0,
                     18.26.4.0/24 1,
                     18.26.7.0/24 2,
                     0.0.0.0/0 18.26.4.1 1);

// Hand incoming IP packets to the routing table.
// CheckIPHeader checks all the lengths and length fields
// for sanity.
```

```
ip :: Strip(14)
    -> CheckIPHeader(18.26.4.255 1.255.255.255)
    -> [0]rt;

c0[2] -> Paint(1) -> ip;
c1[2] -> Paint(2) -> ip;

// IP packets for this machine.
// ToLinux expects Ethernet packets, so cook up a fake header.
rt[0] -> EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> tol;

// These are the main output paths; we've committed to a
// particular output device.
// Check paint to see if a redirect is required.
// Process record route and timestamp IP options.
// Fill in missing ip_src fields.
// Discard packets that arrived over link-level broadcast or multicast.
// Decrement and check the TTL after deciding to forward.
// Fragment.
// Send outgoing packets through ARP to the interfaces.
rt[1] -> DropBroadcasts
    -> cp1 :: PaintTee(1)
    -> gio1 :: IPGWOptions(18.26.4.24)
    -> FixIPSrc(18.26.4.24)
    -> dt1 :: DecIPTTL
    -> fr1 :: IPFragmenter(1600)
    -> [0]fake_arpq0;
rt[2] -> DropBroadcasts
    -> cp2 :: PaintTee(2)
    -> gio2 :: IPGWOptions(18.26.7.1)
    -> FixIPSrc(18.26.7.1)
    -> dt2 :: DecIPTTL
    -> fr2 :: IPFragmenter(300)
    -> [0]fake_arpq1;

// DecIPTTL[1] emits packets with expired TTLs.
// Reply with ICMPs. Rate-limit them?
dt1[1] -> ICMPError(18.26.4.24, 11, 0) -> [0]rt;
dt2[1] -> ICMPError(18.26.4.24, 11, 0) -> [0]rt;

// Send back ICMP UNREACH/NEEDFRAG messages on big packets with DF set.
// This makes path mtu discovery work.
fr1[1] -> ICMPError(18.26.7.1, 3, 4) -> [0]rt;
fr2[1] -> ICMPError(18.26.7.1, 3, 4) -> [0]rt;
```

```
// Send back ICMP Parameter Problem messages for badly formed
// IP options. Should set the code to point to the
// bad byte, but that's too hard.
gio1[1] -> ICMPError(18.26.4.24, 12, 1) -> [0]rt;
gio2[1] -> ICMPError(18.26.4.24, 12, 1) -> [0]rt;

// Send back an ICMP redirect if required.
cp1[1] -> ICMPError(18.26.4.24, 5, 1) -> [0]rt;
cp2[1] -> ICMPError(18.26.7.1, 5, 1) -> [0]rt;

// Unknown ethernet type numbers.
c0[3] -> Print(Bad_Ethernet_Type) -> Discard;
c1[3] -> Print(Bad_Ethernet_Type) -> Discard;
```

ANNEXE C
Fichier de configuration pour IPsec

```
// ipv4_ipsec.click
// Use this file with send-real-Workload.click

// This file is a network-independent version of the IP router
// configuration taken on www.pdocs.lcs.mit.edu/click/
//
// We use an SpdSource and an SpdSink click elements to transfer
// packets from the platform to the interface (SPD)

// eth0, 00:00:C0:AE:67:EF, 18.26.4.24
// eth1, 00:00:C0:4F:71:EF, 18.26.7.1

// Outputs of the Classifiers:
// 0. ARP queries
// 1. ARP replies
// 2. IP
// 3. Other
// We need separate classifiers for each interface because
// we only want proxy ARP on eth0.
c0 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -);
c1 :: Classifier(12/0806 20/0001,
                 12/0806 20/0002,
                 12/0800,
                 -);

Idle -> [0]c0;                  // Just use the ETH1 for the moment

SpdSource(ADDR 0x6fff0000)      //
    -> [0]c1;

out0 :: SpdSink(ADDR 0x6fff0000); // Should use two SPD for each ETH (0 and 1)
out1 :: SpdSink(ADDR 0x6fff0000);
out2 :: SpdSink(ADDR 0x6fff0000); // Packet for encryption

to1 :: Discard;                 // to1 means to Linux or to the control plan.

// An "ARP querier" for each interface.
```

```
fake_arpq0 :: EtherEncap(0x0800, 00:00:c0:ae:67:ef, 00:00:c0:4f:71:ef);
//ARPQuerier(18.26.4.24, 00:00:C0:AE:67:EF);
fake_arpq1 :: EtherEncap(0x0800, 00:00:c0:4f:71:ef, 00:00:c0:4f:71:ef);
//ARPQuerier(18.26.7.1, 00:00:C0:4F:71:EF);
fake_arpq2 :: EtherEncap(0x0800, 00:00:c0:4f:71:ef, 00:00:00:01:02:03);

// Deliver ARP responses to ARP queriers as well as Linux.
t :: Tee(3);                    // Tee(2) should be enough !!!
c0[1] -> t;
c1[1] -> t;
t[0] -> Discard;
t[1] -> fake_arpq0; // was -> [1]arpq0
t[2] -> fake_arpq1; // was -> [1]arpq1

// Connect ARP outputs to the interface queues.
fake_arpq0 -> out0;
fake_arpq1 -> out1;
fake_arpq2 -> out2;

// Proxy ARP on eth0 for 18.26.7, as well as cone's IP address.
ar0 :: ARPResponder(18.26.4.24 00:00:C0:AE:67:EF,
                    18.26.7.0/24 00:00:C0:AE:67:EF);
c0[0] -> ar0 -> out0;

// Ordinary ARP on eth1.
ar1 :: ARPResponder(18.26.7.1 00:00:C0:4F:71:EF);
c1[0] -> ar1 -> out1;

// IP routing table. Outputs:
// 0: packets for this machine.
// 1: packets for 18.26.4.
// 2: packets for 18.26.7.
// All other packets are sent to output 1, with 18.26.4.1 as the gateway.
rt :: StaticIPLookup(18.26.4.24/32 0,
                     18.26.4.255/32 0,
                     18.26.4.0/32 0,
                     18.26.7.1/32 0,
                     18.26.7.255/32 0,
                     18.26.7.0/32 0,
                     18.26.4.0/24 1,
                     18.26.7.0/24 2,
//                   0.0.0.0/0 18.26.4.1 1);
                     0.0.0.0/0 1);
```

```
// Hand incoming IP packets to the routing table.
// CheckIPHeader checks all the lengths and length fields
// for sanity.
ip :: Strip(14)
    -> CheckIPHeader(18.26.4.255 1.255.255.255)
    -> [0]rt;

c0[2] -> Paint(1) -> ip;
c1[2] -> Paint(2) -> ip;

// IP packets for this machine.
// ToLinux expects ethernet packets, so cook up a fake header.
rt[0] -> EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> tol;

// These are the main output paths; we've committed to a
// particular output device.
// Check paint to see if a redirect is required.
// Process record route and timestamp IP options.
// Fill in missing ip_src fields.
// Discard packets that arrived over link-level broadcast or multicast.
// Decrement and check the TTL after deciding to forward.
// Fragment.
// Send outgoing packets through ARP to the interfaces.
rt[1] -> DropBroadcasts
    -> cp1 :: PaintTee(1)
    -> gio1 :: IPGWOptions(18.26.4.24)
    -> FixIPSrc(18.26.4.24)
    -> dt1 :: DecIPTTL
    -> fr1 :: IPFragmenter(1600)      // Put > 1518 if no Fragments wanted
// Added stuff checking if encryption is needed, based on the destination
// address.
// IP routing table emulating an SPD (Security Policy Database). Outputs:
// 0: packets that don't need encryption
// 1: packets to be encrypted with 18.26.4.1 as the gateway
    -> SAD :: StaticIPLookup(18.26.4.0/24 0,
                        0.0.0.0/0 18.26.4.1 1);
SAD[0] -> [0]fake_arpq0;
SAD[1] -> IPsecESPEncap(0x00000001)
    -> IPsecDES(1, 0123456789abcdef, 0)       // Des element
// see p. 23 to see the real fields of the new IP Header
// Here just some dummy values
    -> IPEncap(50, 18.26.4.24, 2.0.0.2)
    -> [0]fake_arpq2;
```

```
// end added stuff...
//      -> [0]fake_arpq0;

rt[2] -> DropBroadcasts
      -> cp2 :: PaintTee(2)
      -> gio2 :: IPGWOptions(18.26.7.1)
      -> FixIPSrc(18.26.7.1)
      -> dt2 :: DecIPTTL
      -> fr2 :: IPFragmenter(300)
      -> [0]fake_arpq1;

// DecIPTTL[1] emits packets with expired TTLs.
// Reply with ICMPs. Rate-limit them?
dt1[1] -> ICMPError(18.26.4.24, 11, 0) -> [0]rt;
dt2[1] -> ICMPError(18.26.4.24, 11, 0) -> [0]rt;

// Send back ICMP UNREACH/NEEDFRAG messages on big packets with DF set.
// This makes path mtu discovery work.
fr1[1] -> ICMPError(18.26.7.1, 3, 4) -> [0]rt;
fr2[1] -> ICMPError(18.26.7.1, 3, 4) -> [0]rt;

// Send back ICMP Parameter Problem messages for badly formed
// IP options. Should set the code to point to the
// bad byte, but that's too hard.
gio1[1] -> ICMPError(18.26.4.24, 12, 1) -> [0]rt;
gio2[1] -> ICMPError(18.26.4.24, 12, 1) -> [0]rt;

// Send back an ICMP redirect if required.
cp1[1] -> ICMPError(18.26.4.24, 5, 1) -> [0]rt;
cp2[1] -> ICMPError(18.26.7.1, 5, 1) -> [0]rt;

// Unknown ethernet type numbers.
c0[3] -> Print(Bad_Ethernet_Type) -> Discard;
c1[3] -> Print(Bad_Ethernet_Type) -> Discard;
```